NATURAL

Natural

User's Guide

Version 5.1.1 for Windows

SOFTWARE AG

This document applies to Natural Version 5.1.1 for Windows and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Table of Contents

User's Guide - Overview . . . 1
User's Guide - Overview . . . 1
Natural Studio - Fundamentals . . . 2
Natural Studio - Fundamentals . . . 2
Starting/Terminating Natural . . . 3
Starting/Terminating Natural . . . 3
Starting Natural . . . 3
Starting a Natural Online Session . . . 3
How to Proceed after Logon . . . 3
Starting a Natural Batch Session . . . 4
Terminating Natural . . . 5
Terminating a Natural Online Session . . . 5
Terminating a Natural Batch Session . . . 5
Configuring Your Natural Environment . . . 6
Configuring Your Natural Environment . . . 6
Changing Default Settings . . . 6
Suppressing the Natural Startup Map . . . 6
Defining Your Own Logon Library . . . 6
Changing the Termination Method . . . 7
Using Objects and Shared Resources . . . 8
Using Objects and Shared Resources . . . 8
What is a Natural Object? . . . 8
Object Types . . . 8
Changing an Object's Type . . . 9
Object Visualisation . . . 10
Object Naming Conventions . . . 10
Object Editors . . . 11
Types of Object Editors . . . 11
Invoking an Object Editor . . . 11
Object Commands . . . 11
Object Operations . . . 13
Creating an Object . . . 13
Copying or Moving Objects - Rules . . . 13
Copying an Object . . . 15
Deleting Objects . . . 15
Exporting Objects . . . 15
Importing Objects . . . 16
Moving Object . . . 16
Listing Objects . . . 17
Printing Objects . . . 17
Renaming an Object . . . 17
Object Retrieval . . . 18
Saving an Object . . . 20
Stowing an Object . . . 20
Shared Resources . . . 21
Using Natural Libraries . . . 22
Using Natural Libraries . . . 22
Logon to a Library . . . 22
Automatic Logon to a Library . . . 22
Manual Logon to a Library . . . 22
Library Types . . . 22
Library Naming Conventions . . . 24
Library Commands . . . 24

Library Operations 24
Creating a New Library 24
Copying or Moving a Library 25
Deleting a Library 25
Renaming a Library 25
Library Limit 26
Example Library for New Features 26
Using Workspace Options 27
Using Workspace Options 27
Setting Workspace Options 27
Display Success Messages 27
Display Delete Messages 27
Display Replace Messages 27
Exit Prompt 28
Show Full Cat All Result List 28
Perform Automatic Refresh 28
Enable Plug-ins 28
Terminal Emulation 28
Using Natural Output Window Options 29
Using Natural Output Window Options 29
Output Window Features 29
Customizing the Mininimize Icon 29
Viewing/Modifying Output Window Profile Settings 30
Output Window General Profile 30
Activate report page buffer 30
Display input fields with frame 30
Display PF-key buttons with number 30
Display more prompt 30
Disable ESC key 30
Disable help menu 30
Fixed fonts only 31
Output Window Color Profile 31
Output Window Font Profile 31
Invoking the Front Profile Dialog 31
Accepting Changes made to the Profile Settings 31
Using Session Parameters 32
Using Session Parameters 32
Setting Session Parameters 32
Session Parameters Grouped by Function 32
Report Parameters 33
Limit Settings 33
Character Assignments 34
Compiler Options 35
Field Appearance 36
Error Handling 37
Data Representation 37
Accessing Tools 38
Accessing Tools 38
Invoking Development Tools 38
Development Tools Available 38
Application Shell 38
Component Browser 38
Debugger 38
Error Messages 38
Frame Gallery 38
Natural Reporter 39

Object Handler . . . 39
Unlock Objects . . . 39
User Exits . . . 39
Arranging Your Natural Environment . . . 40
Arranging Your Natural Environment . . . 40
Displaying the Natural Version of Your Environment . . . 40
Working with Dockable/Floatable and MDI Windows . . . 40
Dockable and Floatable Windows . . . 40
MDI Windows . . . 41
Using Online Help . . . 42
Using Online Help . . . 42
Starting Online Help . . . 42
Displaying the System Command List . . . 42
Displaying Error Message Texts . . . 42
Natural Studio - Introduction . . . 43
Natural Studio - Introduction . . . 43
Natural Studio - Features and Components . . . 44
Natural Studio - Features and Components . . . 44
Natural Studio - Features . . . 44
Natural Studio - Main Components . . . 46
Library Workspace . . . 47
Library Workspace . . . 47
Switching the Library/Application Workspace On or Off . . . 47
Local Environment . . . 47
Remote Environment . . . 47
Logical View . . . 48
Flat View . . . 49
File View . . . 50
Multiple Document Interface Area . . . 51
Multiple Document Interface Area . . . 51
List View . . . 51
Opening a List View . . . 52
List View Operations . . . 52
Editors Available in MDI Area . . . 52
MDI Window Navigation Accelerators . . . 52
Toolbars and Menus . . . 53
Toolbars and Menus . . . 53
Introduction . . . 53
Hiding a Toolbar . . . 54
Positioning a Toolbar . . . 54
Results Window . . . 55
Results Window . . . 55
Switching the Results Window On and Off . . . 55
Using the Results Window . . . 55
Results Interface . . . 56
Results Interface . . . 56
Purpose of the Results Interface . . . 56
Results Window Control Bar Access . . . 56
Tab Handling . . . 57
Image Handling . . . 57
Context-Menu Handling . . . 57
Command Handling . . . 58
Column Handling . . . 58
Row Handling . . . 58
Data Handling . . . 59
Selection Handling . . . 59

Command Line 60
Command Line 60
Using the Command Line 60
Display or Hide the Command Line 60
Associate an Object with the Command Line 60
Status Bar 62
Status Bar 62
Purpose of the Status Bar 62
Switching the Status Bar On or Off 62
Status Bar Option 62
Context Menus 63
Context Menus 63
Accelerators 64
Accelerators 64
Accelerators Grouped by Categories 64
Generally Available Accelerators 64
Dialog Editor Accelerators 65
Program Editor Accelerators 67
Data Area Editor Accelerators 67
Alphabetical Overview of Accelerators 68
Commands 71
Commands 71
Purpose of Natural Commands 71
Issuing Natural Commands 71
Manual Refresh 72
Manual Refresh 72
Purpose of a Manual Refresh 72
Performing a Manual Refresh 72
Customizing 73
Customizing 73
Commands 74
Commands 74
Editing a Toolbar 74
Adding User-Defined Commands to a Toolbar 74
Removing Commands from a Toolbar 74
Rearranging Commands in the Toolbar 74
Toolbars 75
Toolbars 75
Invoking the Toolbars Dialog Box 75
Creating a Toolbar 75
Selecting a Toolbar 75
Removing a Toolbar 75
Keyboards 76
Keyboards 76
Changing Accelerator Key Assignments 76
Removing a Key Assignment 76
Resetting Your Personal Key Assignments 76
User Commands 77
User Commands 77
Invoking the User Commands Dialog 77
Defining a User Command 77
Adding a User Command to an Existing Toolbar 77
Tutorial - Getting Started with Natural 79
Tutorial - Getting Started with Natural 79
Session 1 - Creating and Modifying a Program 79
Session 2 - Creating and Editing a Map 82

Session 3 - Checking and Running a Program 92
Session 4 - Creating a Local Data Area 96
Session 5 - Creating a Global Data Area 103
Session 6 - Creating an External Subroutine 109
Session 7 - Invoking a Subprogram 114
Class Builder 122
Class Builder 122
Introduction 122
What is the Class Builder? 123
Which Classes can be handled by the Class Builder? 123
When is a Class saved? 125
Class Comments 125
Class Builder Interface 126
Logical View 127
Flat View 132
Class Builder Nodes 135
Class Nodes 136
Object Data Nodes 138
GUID Local Data Nodes 139
Interface Nodes 141
Property Nodes 142
Method Nodes 143
Method Implementation Nodes 144
Method Parameter Data Nodes 145
Node Properties 147
General 147
Comments 150
Identification 151
Settings 152
Definition 153
Adding Class Components 154
Link 154
New 155
New Class 155
New Object Data Area 157
New Interface Module 157
New Interface 157
New Method 158
New Property 158
Renaming Class Components 160
Removing Class Components 161
Unlink 161
Delete 162
Editing Class Components 163
Classes 163
Natural Objects 163
Other Class Components 163
Using Interfaces from Several Classes 164
Creating a new Interface Module 164
Linking an existing Interface Module 165
Unlinking an Interface Module 166
Interface Nodes 166
Locking Concept 167
Locking of Classes 167
Locking of Interface Modules 167
Tutorial 168

New class . . . 168
Linking Object Data . . . 168
Creating an Interface . . . 169
Creating Methods . . . 169
Creating Properties . . . 169
Using an Interface Module . . . 169
Linking a GUID Local Data Area . . . 170
Activation Policy . . . 170
Save and Stow Class . . . 171
Register . . . 171
Glossary . . . 172
External Interface . . . 172
Internal Interface . . . 172
Method Implementation . . . 172
Property Implementation . . . 172
Program Editor . . . 173
Program Editor . . . 173
Modifying Program Contents . . . 173
Selecting Text . . . 173
Copying Text . . . 174
Cutting Text . . . 174
Pasting Text . . . 175
Deleting Text . . . 175
Undoing/Redoing Text Changes . . . 176
Renumbering a Program . . . 176
Finding Source Contents . . . 177
Searching for Source Text . . . 177
Searching for a Line Number . . . 179
Editing/Listing Referenced Natural Objects . . . 180
Splitting the Editor Window . . . 180
Jumping between Split Screens . . . 180
Expanding/Collapsing Object Listings . . . 181
Making Listings Expandable/Collapsible . . . 181
Collapsing and Expanding Program Structures . . . 182
Additional Information . . . 182
Recording/Replaying Keystrokes . . . 183
Using Context-Sensitive Help . . . 184
Syntax Help . . . 184
Setting Editor Options . . . 185
Syntax Coloring . . . 187
Font Definition . . . 187
Status Bar Information . . . 188
Map Editor . . . 189
Map Editor . . . 189
Inserting Map Fields and Menus . . . 189
Modifying Map Contents . . . 191
Selecting Fields . . . 191
Deselecting Fields . . . 191
Copying Fields . . . 192
Cutting Fields . . . 192
Pasting Fields . . . 192
Deleting Fields . . . 193
Moving Fields . . . 193
Resizing Fields . . . 193
Aligning Fields . . . 193
Importing Fields . . . 193

Importing System Variables . . . 194
Keyboard Equivalents . . . 195
Defining Fields . . . 196
Defining Text Fields . . . 196
Defining Data Fields . . . 197
Defining Selection Boxes . . . 198
Defining Constant Selection Box Items . . . 199
Defining Variable Selection Box Items . . . 199
Defining a Radio Button . . . 200
Defining a Toggle Button . . . 201
Defining Menu Items . . . 201
Adding a Menu Item . . . 202
Adding a Menu Separator . . . 203
Modifying a Menu Item . . . 203
Removing a Menu Item . . . 204
Defining Push Buttons . . . 204
Defining Bitmaps . . . 205
Defining Field Attributes . . . 206
Defining an Array . . . 208
Modifying Field Colors and Representation . . . 210
Using Field Rules . . . 210
Creating Field Rules . . . 211
Copying Field Rules . . . 211
Editing Field Rules . . . 212
Changing Field Rule Rank . . . 212
Unlinking or Deleting Field Rules . . . 212
Defining Free Predict Rules . . . 213
Converting Free Rules to Inline Rules . . . 213
Defining Key Rules . . . 214
Defining Data Areas for Maps . . . 215
Defining a Data Element . . . 215
Modifying a Data Element . . . 215
Removing a Data Element . . . 216
Testing Maps . . . 216
Previewing Maps . . . 216
Flipping Maps . . . 216
Modifying the Map Profile . . . 217
Setting Editor Options . . . 220
Status Bar Information . . . 220
Data Area Editor . . . 221
Data Area Editor . . . 221
General Information . . . 221
Column Header . . . 222
Generating Copycode from a Data Area . . . 223
Modifying Data Definitions . . . 223
Modifying Fields . . . 223
Copying Fields . . . 224
Cutting Fields . . . 224
Pasting Fields . . . 224
Deleting Fields . . . 225
Inserting Fields . . . 225
Inserting a Data Field . . . 226
Defining an Array . . . 226
Initializing the Value for a Data Field . . . 227
Initializing Values in Free-form Mode . . . 227
Inserting a Global Unique ID . . . 228

Inserting a Constant 228
Inserting a Data Block 229
Inserting a Data Structure 229
Inserting a Handle 230
Searching for Data Fields 233
Setting Editor Options 234
Navigating 234
DDM Editor 235
DDM Editor 235
DDMs Window 235
Adding DDMs 236
Adding DDMs from Adabas Databases 236
Adding DDMs from SQL Databases 236
Additional Options for VSAM Files 237
Data Conversion 238
Modifying DDM Contents 245
Selecting Fields 245
Selecting Attributes in Fields 245
Copying Fields 246
Cutting Fields 246
Pasting Fields 246
Deleting Fields 247
Inserting Fields 247
Modifying DDM Fields 248
Modifying Extended Attributes under a Local Enviroment 248
Modifying Extended Attributes under a Remote Enviroment 248
Displaying Descriptor Information 250
Modifying a DDM Header 250
Modifying Coupling Information 250
Searching for DDM Fields 251
Editor Window Layout 253
Setting Editor Options 253
DDM Editor Options 253
Status Bar Information 253
Column Header 255
Dialog Editor 257
Dialog Editor 257
General Information 257
Dialog Editor Window 258
Changing the Initial Position of the Dialog 259
Changing the Initial Size of the Dialog 259
Selecting/Deselecting Dialog Elements 259
Aborting Mouse Operations 259
Creation Mode in Map Editor and Dialog Editor 260
Changing the Position of a Dialog Element 260
Changing the Size of a Dialog Element 260
Moving the Pointer 260
Simulating the Mouse with the Spacebar 261
Opening Windows and Dialog Boxes Using The Keyboard 262
Scrolling in a Dialog 262
Using the Clipboard 262
Editing Dialogs 263
Editing a Dialog’s Source Code 263
Editing a Dialog’s Attributes 264
Editing a Dialog’s Event Handlers 264
Defining a Dialog’s Menu Bar 265

Defining a Dialog’s Toolbar 265
Creating and Maintaining Timers for a Dialog 265
Adding a Comment Section to a Dialog 266
Defining a Parameter or Local Data Area for a Dialog 266
Selecting a Global Data Area for a Dialog 266
Defining an Inline Subroutine for a Dialog 267
Defining the Control Sequence in a Dialog 267
Dialog Wizard 267
Creating Dialog Elements 269
Importing Data Fields 270
Editing Dialog Elements 271
Cutting a Dialog Element 271
Copying a Dialog Element 271
Pasting a Dialog Element from the Clipboard 271
Deleting a Dialog Element 272
Selecting all Dialog Elements with the same Parent in a Dialog 272
Editing a Dialog Element’s Attributes 273
Editing a Dialog Element’s Event Handlers 273
Unifying the Size of Several Dialog Elements 273
Aligning the Position of Several Dialog Elements 274
Unifying the Spacing Between Several Dialog Elements 274
Stretching a Dialog Element 274
Organizing An Application’s Help File 275
Using The Help Organizer’s Main Dialog 277
Generating Help IDs 279
Extending A Help ID Definition 279
Editing The Global Topic List 280
Setting Editor Options 281
Enabling The Enhanced Listing Option 281
Displaying Or Hiding The Status Bar 282
Turning The Crosshair Cursor On And Off 282
Turning Autoscroll On And Off 282
Displaying The Dialog Inside Or Outside The Editor 283
Displaying Bitmaps 283
Displaying Or Hiding The Grid 283
Customizing The Grid 284
Saving Editor Options With A Particular Dialog 284
Attributes Windows for Dialogs and Dialog Elements 285
ActiveX Control Attributes Window 285
ActiveX Control Property Pages 288
Bitmap Control Attributes Window 288
Canvas Control Attributes Window 290
Control Box Control Attributes Window 292
Dialog Attributes Window 293
Dialog Context Menus Window 296
Edit Area Control Attributes Window 297
Group Frame Control Attributes Window 299
Input Field Control Attributes Window 300
List Box Control Attributes Window 302
Menu Editor Window 304
OLE Container Control Attributes Window 307
Selecting an OLE Server or Document 308
Push Button Control Attributes Window 312
Radio Button Control Attributes Window 314
Scrollbar Control Attributes Window 316
Selection Box Control Attributes Window 318

Signal Attributes Window . . . 320
Status Bar Control Attributes Window . . . 322
Status Bar Control Attributes Subwindow . . . 324
Table Attributes Window . . . 326
Table Attributes Subwindow . . . 328
Text Constant Control Attributes Window . . . 330
Timer Attributes Window . . . 331
Toggle Button Control Attributes Window . . . 332
Toolbar Attributes Window . . . 334
Tool Bar Control Attributes Window . . . 335
Tool Bar Control Attributes Subwindow . . . 338
Dialog Boxes . . . 340
Array . . . 340
Data Area - Local, Parameter . . . 340
Data Area - Global . . . 341
Dialog Compile Error . . . 341
Events . . . 342
Import Data Field . . . 344
Font . . . 344
Source . . . 345
Subroutines . . . 346
Component Browser . . . 347
Component Browser . . . 347
Introduction . . . 347
User Interface . . . 347
Tree View . . . 347
Order . . . 348
Data View . . . 349
Interaction Tree View and Data View . . . 350
Group . . . 350
ActiveX Component . . . 350
Interface . . . 352
Menu . . . 356
Application Development Support . . . 357
Example Construction . . . 357
Automation Objects . . . 362
Interfaces . . . 365
Plug-In Manager . . . 367
Plug-In Manager . . . 367
Purpose of the Plug-In Manager . . . 367
Enabling Plug-In Activation . . . 367
Activating the Plug-In Manager . . . 367
Deactivating the Plug-In Manager . . . 368
Using the Plug-In Manager . . . 368
Natural Studio Sample Plug-In . . . 368
Purpose of the Natural Studio Sample Plug-In . . . 369
Activating the Sample Plug-In . . . 369
Using the Sample Plug-In . . . 369
Deactivating the Sample Plug-In . . . 369
Large and Dynamic Variables/Fields . . . 370
Large and Dynamic Variables/Fields . . . 370
Introduction . . . 370
Definition of Dynamic Variables . . . 370
System Variable *LENGTH - field . . . 371
Size Limitation Checks . . . 371
Statements EXPAND and REDUCE . . . 371

Usage of Dynamic Variables . . . 372
Assignments with Dynamic Variables . . . 372
Initialization of Dynamic Variables . . . 375
String Manipulation with Dynamic Alpha Variables . . . 375
Logical Condition Criterion - LCC - with Dynamic Variables . . . 376
Parameter Transfer with Dynamic Variables . . . 378
Work File Access with Large and Dynamic Variables . . . 382
DDM Generation and Editing for Varying Length Columns . . . 382
Accessing Large Database Objects . . . 383
Performance Aspects with Dynamic Variables . . . 385
Introduction to Event-Driven Programming . . . 386
Introduction to Event-Driven Programming . . . 386
What is an Event-Driven Application? . . . 386
Program-Driven Applications . . . 387
Event-Driven Applications . . . 387
What is Happening Here? . . . 388
Writing Event-Driven Code . . . 388
Components of an Event-Driven Application . . . 388
GUI Development Environments . . . 390
GUI Design Tips . . . 391
Do Your Research . . . 392
Screen Design . . . 392
Menu Design . . . 393
Color Usage . . . 394
Consistency Check . . . 394
Tasks Involved in Creating an Application . . . 395
Tutorial - Overview . . . 396
Creating a Dialog . . . 397
Assigning Attributes to the Dialog . . . 398
Creating Dialog Elements Inside the Dialog . . . 399
Assigning Attributes to the Dialog Elements . . . 401
Creating the Application’s Local Data Area . . . 402
Attaching Event Handler Code to the Dialog Element . . . 402
Checking, Stowing and Running the Application . . . 403
Basic Terminology . . . 404
Event-Driven Programming Techniques . . . 407
Event-Driven Programming Techniques . . . 407
How To Open and Close Dialogs . . . 408
Opening a Dialog . . . 408
Processing Steps When Opening a Dialog . . . 411
Closing Dialogs . . . 411
Initializing Attribute Values . . . 412
How To Edit a Dialog’s Enhanced Source Code . . . 413
What Is The Enhanced Source Code Format ? . . . 413
Avoiding Incompatibilities Between Dialog Editor And Program Editor . . . 414
How To Use The Enhanced Source Code Format . . . 415
How Dialogs, Controls and Items Are Related Hierarchically . . . 416
How To Define Dialog Elements . . . 418
HANDLE OF GUI . . . 418
NULL-HANDLE . . . 419
How To Manipulate Dialog Elements . . . 420
Querying, Setting and Modifying Attribute Values . . . 421
Restrictions . . . 421
Numeric/Alphanumeric Assignment . . . 422
How To Create and Delete Dialog Elements Dynamically . . . 424
Global Attribute List . . . 424

Creating Dialog Elements Statically and Dynamically 424
How to Handle Events of Dynamically Created Dialog Elements 425
How To Enable and Disable Dialog Elements 427
Defining and Using Context Menus 427
Construction 427
Association 428
Invocation 429
Sharing of Context Menus 431
System Variables 432
Generated Variables 432
#DLG$PARENT 432
#DLG$WINDOW 433
Message Files and Variables as Sources of Attribute Values 434
Triggering User-Defined Events 434
Passing Parameters to the Dialog 435
Suppressing Events 437
Menu Structures, Toolbars and the MDI 438
Creating a Menu Structure 438
Parent-Child Hierarchy in Menu Structures 439
Creating a Toolbar 439
Sharing Menu Structures, Toolbars and DILs (MDI Application) 440
Executing Standardized Procedures 442
PROCESS GUI Statement 442
Linking Dialog Elements to Natural Variables 444
Validating Input in a Dialog Element 446
Storing and Retrieving Client Data for a Dialog Element 447
Creating Dialog Elements on a Canvas Control 449
Working with ActiveX Controls 453
Using The PROCESS GUI Statement 460
Working with Arrays of Dialog Elements 464
Working with Control Boxes 466
Purpose of exclusive control boxes 466
Examples of use of exclusive control boxes 466
Creation of the wizard pages 469
Switching between the wizard pages at edit-time 469
Creating the divider line 469
Implementing the ’Back’ and ’Next’ push buttons 470
Clearing all controls on a wizard page 470
Example 2 - a tabbed dialog 471
Working with Error Events 472
Working with a Group of Radio-Button Controls 472
Working with List-Box Controls and Selection-Box Controls 473
Working with Nested Controls 475
Which control types can be containers? 475
Creating a nested control 476
Multiple selection, control sequence and clipboard operations 476
Working with a Dynamic Information Line 478
Working with a Status Bar 479
Working with Status Bar Controls 479
Creating a Status Bar Control 479
Using status bar controls without panes 480
Stretchy vs. non-stretchy panes 480
Outputting text to a status bar control 480
Sharing a status bar in an MDI applications 481
Pane-specific context menus 481
Working with Dynamic Information Line and Status Bar 482

Adding a Maximize/Minimize/System Button . . . 484
Defining Color . . . 484
Adding Text in a Certain Font . . . 486
Adding Online Help . . . 487
Defining Mnemonic and Accelerator Keys . . . 489
Defining a Mnemonic Key . . . 489
Defining an Accelerator Key . . . 489
Displaying Accelerator Keys in Menus . . . 490
Dynamic Data Exchange - DDE . . . 491
Concepts . . . 491
Developing a DDE Server Application . . . 492
Developing a DDE Client Application . . . 492
Return Codes . . . 493
Object Linking and Embedding - OLE . . . 495
What Is OLE In The Natural Context? . . . 495
The OLE Container Control . . . 496
SYSMAIN Utility - Overview . . . 500
SYSMAIN Utility - Overview . . . 500
SYSMAIN Command . . . 500
Copy Object . . . 501
Available XREF Options . . . 501
Move Object . . . 502
Rename Object . . . 503
Delete Object . . . 504
List an Object . . . 504
Find Object . . . 505
Importing Objects to a Library . . . 505
Invoking SYSMAIN by a Subprogram . . . 506
Direct Commands . . . 506
Additional Keywords for Direct Commands . . . 509
Tamino Server Extensions . . . 510
Tamino Server Extensions . . . 510
Introduction . . . 510
Overview . . . 510
Developing a Tamino Server Extension . . . 511
Overview . . . 511
Set the Library SYSEXSXS as Steplib . . . 511
Create a New Library for Your Project . . . 511
Create a NaturalX Class . . . 512
Create the Object Data Area . . . 512
Edit the Object Data Area . . . 512
Link the Connection Interface . . . 512
Implement the Method Connect . . . 513
Add Server Extension Functions . . . 513
Save and Catalog the Class . . . 515
Using Callbacks . . . 515
Deploying a Tamino Server Extension . . . 516
Installing a Tamino Server Extension . . . 516
Tamino Server Extension Sample . . . 516
XML Toolkit . . . 518
XML Toolkit . . . 518

User's Guide - Overview

The Natural User's Guide describes features of the Natural Studio that you will use on a daily basis to create and maintain applications. First, you are introduced to the landscape of the graphical user interface so that you always find what you need exactly when you need it. Then you are provided with a task-oriented description of each of the major editors and tools used to create applications: the Program Editor, Data Area Editor, Map Editor, DDM Editor and Dialog Editor, the Class Builder and the Component Browser.

Tutorials are provided to demonstrate how Natural applications can be structured and how to get the most out of the features of each tool. Note that it is outside the scope of this documentation to provide you with a comprehensive methodology for developing applications. This documentation cannot replace a hands-on training course in Natural programming.

- Natural Studio - Fundamentals
- Natural Studio - Introduction
- Tutorial - Getting Started with Natural
- Class Builder
- Program Editor
- Map Editor
- Data Area Editor
- DDM Editor
- Dialog Editor
- Component Browser
- Plug-In Manager
- Introduction to Event-Driven Programming
- Event-Driven Programming Techniques
- SYSMAIN Utility
- Tamino Server Extensions
- XML Toolkit

Readers of this documentation are assumed to have a fundamental working knowledge of Microsoft Windows and the terminology used to describe it. If not, consult the Windows documentation for a description of basic Windows elements, usage and terminology.

Natural Error Messages

Natural error messages can be retrieved online. You can open the menu "Help - Natural Errors" in the menu bar to get the help on errors dialog or you can type the help command into the command line.

Related Topics

- Application Shell
- Frame Gallery

Natural Studio - Fundamentals

- Starting/Terminating Natural
- Configuring Your Natural Environment
- Using Objects and Shared Resources
- Using Natural Libraries
- Using Workspace Options
- Using Natural Output Window Options
- Using Session Parameters
- Accessing Tools
- Arranging Your Natural Environment
- Using Online Help

See also:

- Natural Studio - Introduction
- Application Shell
- Frame Gallery

Starting/Terminating Natural

The following topics are covered below:

- Starting Natural
  - Starting a Natural Online Session
  - How to Proceed after Logon
  - Starting a Natural Batch Session
- Terminating Natural
  - Terminating a Natural Online Session
  - Terminating a Natural Batch Session

Note: As a prerequisite, general knowledge of how to use the mouse and the desktop is required to use this product. If you are not sure how to use the mouse and/or desktop, please refer to the documentation of your operating system in use.

See also:

- Natural Execution Configuration (for information on the relevant parameters in the NATPARM module)
- Profile Parameters (Detailed Descriptions)
- Natural Configuration Utility (Usage)

Starting Natural

Starting a Natural Online Session

▶ **To start a Natural session from your desktop**

- From the **Start** menu, choose **Programs** > **Software AG Natural** ***v.r.s***> **Natural**.
  - where *v.r.s.* is the Natural *version*, *release*, *system maintenance level* number.

While the program is loaded, the Natural startup map is displayed informing you about the loading in progress. See also Suppressing the Natural Startup Map.

After logon, Natural Studio, the development environment for Natural, appears.

How to Proceed after Logon

When you start Natural for the very first time, the library workspace will be displayed on the left side of the development environment with the current logon library selected. When Natural is restarted, the toolbars and windows are placed at the same positions as when Natural was left the last time.

For details on how to proceed with Natural Studio or the Natural Configuration Utility, refer to:

- Natural Studio - Features
- Natural Studio - Main Components
- Natural Configuration Utility
- Changing Default Settings
- Defining Your Own Logon Library

Starting a Natural Batch Session

You can start Natural in batch mode from the "Target" text box of the "Properties" of the Natural program-item icon. For this purpose, it is currently necessary to create an additional Natural icon on your desktop, following the procedure described below. (If you just copy an existing Natural program-item icon, the "Target" field may not work..)

1. Use the **New** > **Shortcut** option of the context menu to create another Natural icon on your desktop.
2. In the command line of the "Create Shortcut" window, specify the path to the natural.exe file, for example:
   `"C:\Program Files\Software AG\Natural\5.1.1\Bin\natural.exe"`
   and follow the instructions.
3. Rename the newly created "natural.exe" icon, for example, to "Natural Batch"
4. Select this new Natural program-item icon
5. Click the right mouse button and choose the **Properties** option from the context menu.
   The Natural Properties dialog box appears.
6. In the "Target" field, add the "batchmode" parameter, for example:
   `"C:\Program Files\Software AG\Natural\5.1.1\Bin\natural.exe"` **`batchmode`**.

To invoke Natural in batch mode from the "Target" text box

- Click to the additional Natural program-item icon.

For information on the batch-mode-relevant profile parameters in the NATPARM module, refer to Batch Mode (in the Natural Operations documentation, under the heading Profile Parameters Grouped by Function).

For special considerations that apply when running Natural in batch mode, see Natural in Batch Mode.

Terminating Natural

Terminating a Natural Online Session

To terminate a Natural online session

- From the **Object** menu, choose **Exit**;
  Or press ALT+F4;
  Or enter the system command "FIN" in the command line;
  Or execute a Natural program that contains a TERMINATE statement.

See also: Changing the Termination Method

Terminating a Natural Batch Session

A Natural batch mode session will be terminated when one of the following is encountered during the session:

- a FIN command in the input dataset,
- an end-of-input condition in the input dataset,
- a TERMINATE statement in a Natural program which is being executed.

Configuring Your Natural Environment

This document summarizes the facilities you have to adapt your Natural environment to your specific needs. The following topics are covered below:

- Changing Default Settings
- Suppressing the Natural Startup Map
- Defining Your Own Logon Library
- Changing the Termination Method

Changing Default Settings

By changing the default settings supplied, you can set up your own user profile. Refer to:

- Profile Parameter Usage
  This document describes how and where you can use Natural profile parameters to define various characteristics of the Natural environment.
- Profile Parameters Grouped by Function
  Lists the individual parameters contained in the NATPARM parameter file (or an alternate parameter file) that can be changed in the Natural Configuration Utility.
- Natural Configuration Utility
  Used to create and modify global and local configuration files and parameter files.
- Configuration Files
  These configuration files should only be modified by a system administrator.

Suppressing the Natural Startup Map

When you specify a colon "**:**" in the target edit control of the Natural short-cut property sheet, the Natural startup map will be suppressed.

Example:
`C:\Program Files\Software AG\Natural\v.r.s\Bin\natural.exe:`
- where *v.r.s.* is the Natural *version*, *release*, *system maintenance level* number.

To get to the target edit control

1. Right click on the Natural icon

   on your desktop.

2. Then choose **Properties**.

Defining Your Own Logon Library

When you first log on to Natural, your logon library will be SYSTEM by default. You can define your own logon library in the Natural environment or use a library set up for you by your administrator.

You can alter your default library from SYSTEM to any other library by setting the Natural profile parameter INIT-LIB in the NATPARM module, using the Natural Configuration Utility.

Changing the Termination Method

Termination methods can also be modified in the NATPARM parameter file in node Program Loading/Deletion, using the Natural Configuration Utility.

Using Objects and Shared Resources

This document describes the use of Natural objects and also the use of Shared Resources (Non-Natural objects). The following topics are covered below:

- What is a Natural Object?
- Object Types
- Object Visualisation
- Object Naming Conventions
- Object Editors
- Object Commands
- Object Operations
- Object Retrieval
- Shared Resources

See also:

Library Limit (for max. number of objects in a library)

What is a Natural Object?

An object in Natural terms is a programming module which is used in creating a Natural application. It is always associated with a specific library. Several object types exist depending on the Natural application task (e.g. a component-based application or an application providing a graphical user interface).

Object Types

Natural provides the following object types:

- **Data area**
  A module containing the descriptions of data to be used by a Natural program (which is a Natural object for itself). It usually contains a declaration of user-defined variables and constants, as well as referenced database fields in the form of data definition modules. Data areas can be global (capable of being shared by two or more Natural programs), local (contained within programs), parameter data areas, or object data areas.
- **Data definition module (DDM)**
  A logical grouping of database fields and field descriptions which makes it possible to access database information from within a program. The data definition module must be referenced in a data area serving the program.
- **Program**
  A set of instructions or actions that are executed procedurally.
- **Subprogram**
  A Natural object that is called by another Natural object. A subprogram can receive parameter values from the Natural object which calls it. Passed parameters must be included in a parameter data area.
- **Subroutine**
  A Natural object that is called by another Natural object. A subroutine can receive parameter values from the calling program. A subroutine has automatic access to the same global data area as the Natural object which calls it.
- **Map**
  A layout for the information provided on screens referenced by another Natural object (for example, in a program) for data input and output. In a map, the end user typically enters the information that is necessary for processing a Natural object.

- **Helproutine**
  A text element that can be assigned to a map or field in order to provide users with context-sensitive help for an application.
- **Copycode**
  A portion of Natural source code which is automatically included in an external program when this program is compiled. Copycode promotes modularization in applications by making it possible for many different programs to use the same module of source code.
- **Text**
  Text can be added and modified with the program editor, and stored in a Natural library using the SAVE command to store it. The naming conventions for Natural object types apply.
- **Dialog**
  Dialogs are used in conjunction with event-driven programming when creating Natural applications for graphical user interfaces (GUIs). For further information, see Event-Driven Programming Techniques.
- **Class**
  (for further information, see NaturalX documentation and the Class Builder).
- **Resource**
  Resources are only available with Natural under Windows.

Changing an Object’s Type

You can change the type of any object except a DDM, map, or dialog. For the remaining objects, the following restrictions apply:

- A data area can be converted only to another type of data area: local, global, or parameter.
- A programming object can be converted only to another type of programming object: program, subprogram, subroutine, helproutine, copycode, text.

To change an object’s type

1. Open the object and then, from the **Object** menu, choose **Object Type**.
   A cascading menu is displayed listing all possible object types for selection.
2. Choose the type of object to convert to.
   The new type is displayed in the title bar. The object must be saved to make the conversion complete.

Object Visualisation

The Natural objects displayed can consist of the Natural source, the Natural generated program or both. The difference is reflected by a bitmap. The green ball always indicates that a generated program is available, without the green ball it is indicated that only the source of the displayed object is available and finally a green ball as a non-greyed bitmap indicates that both a source and a generated program is available.

Object Naming Conventions

Length

The name of a Natural object can be 1 to 8 characters long.

Structure

The name of a Natural object can consist of the following characters:

Character	Explanation
A - Z	upper case alphabetical characters
0 - 9	numeric characters
-	hyphen
@	commercial "at" sign
_	underline
/	slash
$	dollar sign
&	ampersand (only as language code character)
#	hash/number sign
+	plus sign (only allowed as first character)

Additional Conditions

- The **first** character of the name must be one of the following:
  - an upper-case alphabetical character (A - Z)
  - a hash/number sign (#)
  - a plus sign (+)
- If the first character is a "#" or "+", the name must contain at least one additional character.

Object Editors

- Types of Object Editors
- Invoking an Object Editor

Types of Object Editors

Natural provides the following object editors:

- Data Area Editor
  Used to create and maintain global data areas, local data areas, parameter data areas, and object data areas. This editor has a column format that is designed for defining the user-defined variables and database fields used in Natural programs, subprograms, subroutines and dialogs.
- DDM Editor
  Used to create and maintain data definition modules (DDMs).
- Program Editor
  Used to create and maintain Natural programs, subprograms, subroutines, helproutines, copycode, text and classes (see also Class Builder).
- Map Editor
  Used to create and maintain maps. Extended field editing features make it possible to assign special properties to fields. Processing rules can be attached to fields in the map.
- Dialog Editor
  Used to create event-driven applications composed of dialogs.
- Class Builder
  Used to create and maintain classes.

Invoking an Object Editor

To invoke an object editor from the tree view

- Double-click the object in the tree view.
  Or click on the corresponding icon in the status bar.

To invoke an object editor from the command line

- Issue the system command EDIT.
  If you select an existing object for editing, the appropriate editor is invoked automatically.

Object Commands

This section describes all of the System Commands that you can perform on Natural objects.

Command	Purpose
EDIT	Edit the source form of an object.
CLEAR	Close the currently active object and open a new editor window that has no content and no name.If this object has been modified since the last save, you are prompted to save any changes.
CHECK	Check the source code of an object for syntax errors. Syntax checking is also performed as part of the RUN and STOW commands.
CATALOG	Cataloging an object compiles the source program in the active editor window and stores the resulting object module. For a full description, see the CATALOG command.
UNCATALOG	Delete one or more generated programs (GPs). An object can only be uncataloged if it has already been cataloged.
SAVE	Save the **source form** of the Natural object currently in the work area of the editor. Syntax is not checked. A saved program can be RUN, but not executed (see EXECUTE command below).
STOW	Save the **source form** of an object, compile the object and store the resulting generated program (GP) as well as the source. The object is syntax-checked during the compilation process.
SCRATCH	Delete the **source and object form** of an object. A list of all objects stored in the current library will be displayed; on the list you can then mark the object(s) to be deleted.
RUN	Compile and execute a source program.
EXECUTE	Execute a program that has been compiled and stored in object form.
DEBUG	Invoke the Natural debugging facility for a cataloged program or dialog. For further information, refer to the Natural Debugging documentation.

Object Operations

The Object Operations such as copy, move, rename, delete, import or export can be applied inside the Library Workspace and inside any open Listview. In contrast to the library workspace where only one object can be selected at a time, multiple objects of a listview can be selected.

It is possible to use these operations in both the Local and Remote environment. The operations can also be used across environments.

The following topics are covered below:

- Creating an Object
- Copying or Moving Objects - Rules
  - Valid Source Nodes
  - Valid Target Nodes
- Copying an Object
- Deleting Objects
- Exporting Objects
- Importing Objects
- Moving Objects
- Listing Objects
- Printing Objects
- Renaming an Object
- Object Retrieval
- Saving an Object
- Stowing an Object

Creating an Object

To create an object using a context menu

- Select a library node, open the context menu, and select **New** with the corresponding object type.
  Or select a group node in the logical view (e.g. "Programs") and choose **New** from the context menu.
  The editor object of the currently selected object type is opened.
  Note:
  New classes can only be created in the Logical or Flat View. For more information, refer to the Class Builder.

To create an object using the menu bar

- From the **Object** menu, choose **New**.
  Except for classes, a specific node needs not to be selected.
  For classes, a library node or the "Classes" group node has to be selected.

To create a program editor object using accelerator keys

- Use **CTRL-N**.

Copying or Moving Objects - Rules

Using Drag and Drop / Cut, Copy and Paste

With these operations, Natural Studio provides a powerful technique for the operations Copy and Move. It is possible to use these operations for nearly all of the available nodes in the library workspace and the various list views.

For example, you can copy all programs of a library by applying the operation on the "Programs" node in the "library workspace"
logical view or you can move all generated programs of a library by applying the operation on the "Gp" node in the "library workspace" file view.

Two important rules have to be taken into account when these operations are used.

- **The target node of a copy/move operation will only accept the objects of the selected source node when actually ALL objects can be copied/moved to the target node.**
  Imagine a library is opened in a flat list view and the "Copy" operation is applied on a couple of selected objects. The "Paste" operation on the "Src" Node in the file view is not allowed (indicated by the greyed "Paste") when the object list contains any Natural generated program since the target only accepts Natural source files.

- **A group node can only be copied to a library, it cannot be copied into another group node.**
  For example, it is not possible to copy the "Programs" node of a library into the "Programs" node of another library, but it is possible to copy the "Programs" node into the "Library" node itself.

Valid Source Nodes

Regarding the previously described rules, the following nodes can be selected to act as a source for Copy or Move operations:

- All group nodes in the logical view ("Programs", "Subprograms", ...).
- All subdirectory nodes in the file view ("Gp", "Src").
- All objects in any view.
- Libraries.

Valid Target Nodes

Regarding the previously described rules the following nodes can be selected to act as a target for Copy or Move operations:

- All group nodes in the logical view ("Programs", "Subprograms", ...).
- All subdirectory nodes in the file view ("Gp", "Src").
- All objects in any view.
- The library nodes in any view.

Copying an Object

To copy an object using the context menu

1. Select a source node.
2. Open the context menu and choose **Copy**.
3. Select a target node, open the context menu and choose **Paste**.

To copy an object using accelerator keys

1. Select a source node and press **CTRL-C**.
2. Then select a target node and press **CTRL-V**.

To copy an object using left mouse button drag & drop

1. Select the source node with the left mouse button and drag it to the target node.
2. Before releasing the mouse button, press **CTRL**.
3. Apply a drop of the source node by releasing the left mouse button.

To copy an object using right mouse button drag & drop

1. Select the source node with the right mouse button and drag it to the target node.
   After the right mouse button has been released, a context menu pops up.
2. Choose **Copy**.

Deleting Objects

It is, for example, possible to delete all Natural objects of type Copycode of a library by deleting the "Copycodes" node of a library in the "library workspace" Logical View or delete all generated programs by deleting the "Gp" node of a library in the "library workspace" File View.

To delete objects

- Select the object and press **DEL**.
  Or choose **Delete** from the context menu.
  Or choose **Delete** from the **Object** menu.

Exporting Objects

Applying the copy and move operations described in the section Copying or Moving Objects, it is also possible to export Natural objects from the Natural environment to the Windows Explorer using Drag & Drop or Cut, Copy and Paste.

This operation can be done for any node except the system file nodes in general ("user libraries", "system libraries").

- When the "Src" or "Gp" node of a library in the "library workspace" File View is exported, the whole directory is copied/moved.
- When a logical group node (e.g. the "Programs" node) of a library in the "library workspace" Logical View is exported, all the individual files of the selected type ("Program"), that means, all files with the extension ".NGP" (Natural Generated Program) and ".NSP" (Natural Source Program) will be copied/moved.
- When a Natural object node (e.g an object of type Dialog named "MYDLG") of a library in the "library workspace" Flat View is exported, the corresponding files will be copied or moved (MYDLG.NS3 if the source is available, MYDLG.NG3 if the corresponding generated program is available).

Importing Objects

Applying the described copy or move operations in the section Copying or Moving Objects, it is also possible to import files from the Windows Explorer to the Natural environment using Drag & Drop or Cut, Copy and Paste.

- If a complete directory is being imported, only objects with a file extension valid to Natural (e.g ".NSP", "NG3..." will be accepted, subdirectories will be ignored.
- If multiple directories are being imported, the same rule for importing a single directory applies.
- If multiple files are being imported, all files with an invalid file extension are ignored.

Note:
Before the import is started, it is important to set the proper mode (structured mode or reporting mode) in which the Natural objects to be imported were developed, otherwise some of them might not be compiled. This can be done using the command `Globals SM=ON/OFF`.

If an object is imported and the object name is unknown to Natural and exists in the library, a container name will be generated with the object name identical plus a running index.

Moving Object

To move an object using the context menu

1. Select a source node and, from the context menu, choose **Cut**.
2. Then select a target node and, from the context menu, choose **Paste**.

To move an object using accelerator keys

1. Select a source node and press **CTRL-X**.
2. Then select a target node and press **CTRL-V**.

To move an object using left mouse button drag & drop

1. Select the source node with the left mouse button and drag it to the target node.
2. Apply a drop by releasing the left mouse button.

To move an object using right mouse button drag & drop

1. Select the source node with the right mouse button, drag it to the target node and release the right mouse button.
2. From the resulting context menu, choose **Move Here** to accomplish the drop.
   After the operation is complete, the source node is deleted from the environment.

Listing Objects

The LIST command is used to display the source code of objects. When an object is displayed using the LIST command, its content can be copied but not modified.

To list an object

- Select the object and choose the List command from the context menu.
  Or click the List button in the object tool bar.

Printing Objects

You can print the source listing of an object. A dialog box is displayed in which you can select the number of copies you want and modify printer defaults. To modify your printer defaults, choose **Print Setup**.

To print an object

- Select the object and, from the context menu, choose **Print**.
  Or click the **Print** button in the **Object** tool bar.
  Or choose **Print** from the **Object** menu.

Printing an Open Object

To print an object that is open on your desktop and active

1. Click the **Print** tool bar button.
2. In the "Copies" text box, enter the number of copies you want to print.

In the "Properties" dialog box, you can specify various page setup and advanced printer settings such as those applicable on the Windows Explorer. If you mark the Print to File option, the document is "printed" to a file whose location will be the current directory, unless you specify a specific directory.

Renaming an Object

Renaming of objects can be accomplished by in-place-editing. Only one object can be renamed at a time. If several nodes in a list view are selected and a **Rename** is applied from the context menu, the operation is performed for the node currently having the focus.

To start in-place-editing

1. Select the node.
2. Press the right mouse button and, from the resulting context menu, choose **Rename**.
   Or press **F2**.
   Or click on the selected node with the left mouse button.
   Or press **ESC** to abort the rename process.

To rename an object from the Object menu

1. Select the object and, from the menu bars's **Object** submenu, choose **Rename**.
2. In the object's name field, enter the new name.
3. Press **ENTER** to finish the in-place-editing process.
   Or click with any mouse button on a different position.
   Or press **ESC** to abort the rename process.

Object Retrieval

With the "Find Objects" dialog, it is possible to find Natural objects and the specified containing text. It can be applied on any node in Natural Studio.

To start the "Find Objects" dialog

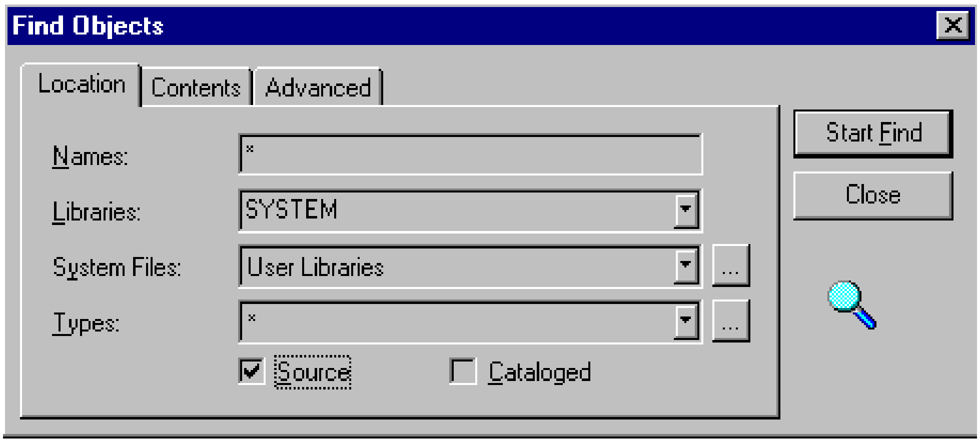
- Open menu **Library - Find Objects** in the Menu bar.
  Or open a context menu for any node in the library workspace.
  Or open a context menu for any node in an active list view

The "Find Objects" dialog comprises the "Location" sheet.

"Location" Sheet

With the "Location" sheet, the following settings can be applied to find Natural objects:

Names	Names of the objects to be found. It is possible to specify multiple names separated with a semicolon. Additionally wildcards can be used.
Libraries	The libraries in which to search for objects; it is possible to specify multiple names separated with a semicolon.
System File	The systemfile to be used for the search.
Types	The types of Natural objects to be included in the search; open the "Types" dialog to select the types.
Source	If this box is checked, a search only for Natural sources is to be performed.
Cataloged	If this box is checked, a search only for Natural generated programs is to be performed.



Names: *
Libraries: SYSTEM
System Files: User Libraries
Types: *
☑ Source □ Cataloged

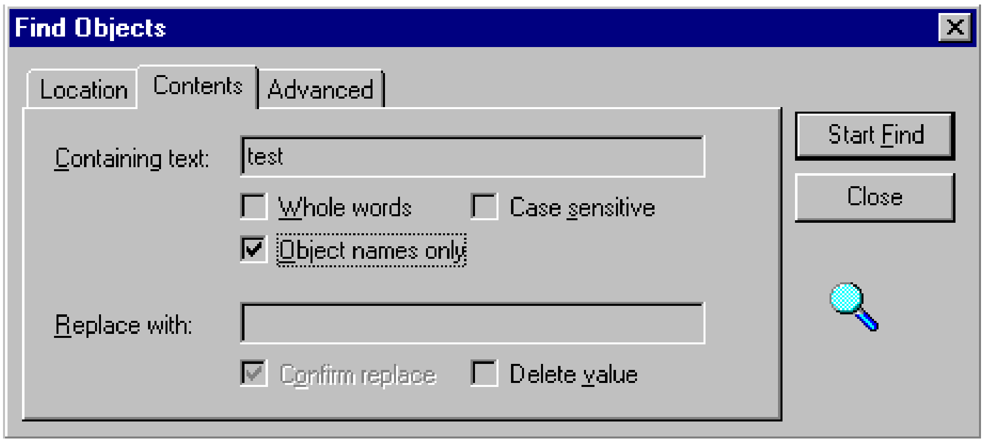
In the above "Location" sheet, a search for any object is started in the "system" library of the "user libraries" system file. Only Source objects are to be included in the search.

Note:
If both the "Source" **and** the "Cataloged" check boxes are checked, both the source and the generated program of the object must exist in order for the object to be found.

"Contents" Sheet

With the "Contents" sheet, it is possible to scan the objects requested in the "Location" sheet for text including an optional replacement of text.

Containing text	The text to be searched for.
Whole words	If this box is checked, only whole words are taken into account.
Case sensitive	If this box is checked, case sensitivity of the text is taken into account.
Object names only	If this box is checked the object where the text is found will only be displayed once in the result list.
Replace with	The text to replace the found text with.
Delete value	If this box is checked, the found text will be deleted. This button is disabled when a "replace" text is specified.
Confirm replace	If this box is checked, a confirmation dialog is displayed for replacing the text.



Containing text: test

☐ Whole words ☐ Case sensitive

☑ Object names only

Replace with:

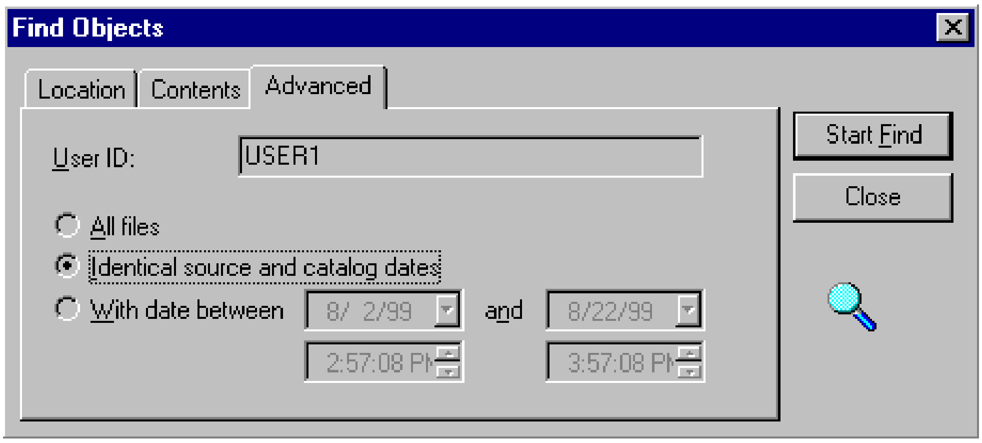
☑ Confirm replace ☐ Delete value

In the above "Contents" sheet the containing text "test" is being searched. If multiple occurences are found in an object, the result list displays the object only once with the first occurrence displayed.

"Advanced" Sheet

With the "Advanced" sheet it is possible to specify additional criteria for finding objects.

User ID	If specified, only objects with a matching user ID will be found.
All files	If this radio button is set, a search in all specified files takes place.
Identical source and catalog dates	If this radio button is set, only objects where the catalog and the source date match are found.
With date between	If this radio button is set, only objects where the last modification date is located between the specified range are found.



User ID: USER1

○ All files
● Identical source and catalog dates
○ With date between 8/ 2/99 and 8/22/99
2:57:08 PM 3:57:08 PM

In the above "Advanced" sheet, only objects which were created with the user ID "USER1" are found. Additionally, the search is only successful when the last modification date of the object's source and generated program match.

Saving an Object

The SAVE command is used to store the source object in the active editor window. The window is not closed afterwards. If no library name is specified in the SAVE command, the object is saved to the library from which it was opened or created, which is not necessarily the active library (the library displayed in the status line).

To save an object:

- From the **Object** menu, choose **Save**.
  Or click the **Save** toolbar button.
  Or enter the SAVE command at the command line.

The source of the object in the active window is saved to the library displayed in the status line.

Saving an Object with Another Name

This function creates a new object by copying the current contents of the editor to a new object and closing the original object. If no modifications have been made to the object since it was last saved, the function operates like a simple copy. If, however, modifications have been made since the last save, the new object contains the modifications and the old object is closed without saving the changes.

To save an object to another name:

1. From the **Object** menu, choose **Save As**.
   Or click the **Save As** toolbar button.
   A dialog is displayed where you can specify a new object name, library, and type.
2. Choose **OK**.

Stowing an Object

The STOW command stores an object in both source and object module form. First the source object is saved, then a syntax-check is performed to determine whether the object can be compiled. If no syntax errors are found, the object is compiled and the resulting object module is stored. The window is not closed afterwards. If no library name is specified in the STOW command, the object is stowed to the library from which it was opened or created, which is not necessarily the active library (the library displayed in the status line).

▶ **To stow an object:**

1. In the "Objects" window, select the object.
   Or open the object.
2. From the **Object** menu, choose **Stow**.
   Or click the **Stow** toolbar button.
   Or enter the STOW command at the command line.

The object code of the object in the active window is stowed to the library displayed in the status line.

Shared Resources

A shared resource is any non Natural object such as a bitmap or a help file and is always associated with a specific library.

Natural Studio provides the same operations on shared resources as those applicable on the Windows Explorer. As with Natural objects, new shared resources can be created inside the Studio, they can be exported or imported, deleted, renamed, copied or moved. Since they are part of a library, they are also included in the library search order defined by the STEPLIB assignments.

For more information on resources, see the Programming Guide - Object Types - Using Non-Natural Files - Resource and User’s Guide - Dialog Editor - ActiveX Control Property Pages.

Using Natural Libraries

A library in Natural terms is the container for Natural objects. A Natural application can access objects in multiple libraries depending on how the environment is set up.

The following topics are covered below:

- Logon to a Library
- Library Types
- Library Naming Conventions
- Library Commands
- Library Operations
- Library Limit
- Example Library for New Features

See also:

- *STEPLIB (for information on the STEPLIB concept)
- Defining Your Own Logon Library

Logon to a Library

In order to work with objects in a Natural library or to start an application inside a specific library, a Logon to this library must first be performed (see also the STEPLIB concept).

Automatic Logon to a Library

As of Natural 5.1.1, an automatic logon is performed any time a different library is selected or an object inside a different library is selected.

Manual Logon to a Library

Apart from the implicitly performed mechanism, it is still possible to perform a manual logon with the LOGON command in the Command Line.

Generally, you can change libraries anywhere in Natural by entering the following system command in the command line:

```
LOGON library-ID
```

where *library-ID* is the ID (name) of the library you want to access.

Library Types

Three types of libraries are available.

System Library	The system libraries are reserved for Software AG purposes only and are subject to change without notice. The currently available system libraries are also known as the FNAT systemfile which is version-dependent. **Important:** Do not put any application-specific data in this library. **Example of a system library:** The library "SYSERR" represents a system application to maintain error messages. A system library is always named with the "SYS" prefix in the library name.
User Library	Used to develop the application. The currently available user libraries are also known as the FUSER systemfile. It contains all the objects of an application which are specific to this application. A user library is always named with the prefix not equal to "SYS" in the library name.
Inactive Library	A library which cannot be accessed by an application. For using an inactive library, the status must be changed from inactive to active, that means the correspondig inactive systemfile must become an active one (either FUSER or FNAT). For more information, refer to FUSER, FNAT and System Files in the Natural Operations documentation. Inactive libraries can be used for the Object Operations and Library Operations only.

Library Naming Conventions

The name of a Natural library can be 1 to 8 characters long. It must start with an upper-case alphabetical character and can consist of the following characters:

Character	Explanation
A- Z	uppe case alphabetical characters
0 - 9	numeric characters
-	hyphen
_	underline

Library Commands

The CATALL command is used to catalog all objects in the current library. For a full description, see the CATALL command.

Library Operations

- Creating a New Library
- Copying or Moving a Library
- Deleting a Library
- Renaming a Library

The library operations like copy, move, rename or delete can be applied inside the library workspace and inside any open systemfile listview.

Creating a New Library

A new library can only be created in a systemfile (FNAT or FUSER) located inside the Local Environment and only one library can be created at a time.

In addition, the following restrictions exist:

- When a new library is to be created in the "user libraries", the library name **must not** start with "SYS".
- When a new library is to be created in the "system libraries", the library name **must** start with "SYS".
- Furthermore, a new library named "SYSTEM" must not be created.

To create a new library

1. Select the corresponding systemfile node (e.g. "user libraries" in the library workspace logical view).
2. From the node’s context menu, choose **New**.
   With this command, an in-place-editing process is started.
   Natural creates a default name sorted into the existing library list which can be changed to any name conforming to the library naming conventions.
3. To finish the in-place editing, press **ESC** or **ENTER**.
   Or click with the mouse button on a different location.

The new library is inserted sorted into all library views.

Copying or Moving a Library

To copy or move a library

1. Select the corresponding library node in the library workspace.
   Or open the Local Environment or Remote Environment list view.
   The target node of the copy or the move of a library can be any systemfile node (FNAT, FUSER or an inactive systemfile) or even any other library node. In the latter case all objects of the source library are copied.
2. Copy a library to the "user libraries" (FUSER) or the "system libraries" (FNAT) systemfile.

When a library is copied to the "user libraries" system file of the **active environment**, the new library must conform to the naming conventions for user libraries (FUSER), that means the library must not start with "SYS". Therefore, a dialog is displayed where the default library name provided ("USRLIB") can be overwritten.

The same handling applies to the system file FNAT where the library name must start with "SYS". In this case, the dialog mentioned offers the default name "SYSLIB".

In all other situations, for example, when copying a library from FNAT to an inactive system file, the name of the source library is taken as the target library name.

Deleting a Library

If you are working in a multiple-user environment, you should only delete a Natural library if you have exclusive access to the library involved.

To delete a library

1. From the **Library** menu, choose **Delete.**
   The "Delete a Library" dialog box is displayed.
2. From the "Library" drop-down list box, select the library to be deleted.
   Deselect the **Confirm on Delete** toggle button to suppress deletion confirmation messages.
   If a confirmation message is desired, leave the toggle button selected.
   Choose **OK** or press **ENTER**.
   After confirmation, the library is deleted.
   Or select the library and press the **DEL** key.
   Or select the **Delete** item from the context menu.
   Or select the **Delete** item from the menu bar.

Renaming a Library

Renaming of a library is done with in-place-editing. Only one library can be renamed at a time.

If several libraries in a list view are selected and the **Rename** item is applied, the operation is performed for the library node currently having the focus.

The name of a user library (FUSER) must not start with "SYS", whereas a system library has to start with "SYS". The library named "SYSTEM" cannot be renamed.

To start in place editing

1. Select the node.
2. Open the context menu and apply the **Rename** item.
   Or press **F2**.

Or click on the selected node with the left mouse button.

3. Enter the new library name and press **ENTER** to confirm (or **ESC** to cancel).
   The in-place-editing process is finished.

Alternative method of doing the rename:

1. Select the item.
2. Choose **Rename** from the menu bar’s **Object** submenu.
3. Enter the new name.
4. Press **ENTER** to confirm (or **ESC** to cancel).
   Or click with any mouse button on a different position.
   The in-place-editing process is finished.

Library Limit

The maximum number of Natural objects that can be contained in a Natural library is 30000.

Example Library for New Features

The library SYSEXV contains several example programs which illustrate some of the new features of Natural version 5.1. and the former version 4.1.

To access the example programs

1. Log on to library SYSEXV and open the folder "Dialog".
2. Execute the dialog VERSION.

A dialog is displayed from which you can select the respective version and its example programs.

Using Workspace Options

The following topics are covered below:

- Setting Workspace Options
- Display Success Messages
- Display Delete Messages
- Display Replace Messages
- Exit Prompt
- Show Full Cat All Result List
- Perform Automatic Refresh
- Enable Plug-ins
- Terminal Emulation

Setting Workspace Options

You may want to prevent certain messages from appearing which are generally used to confirm that a particular action has been or will be carried out.

To display the workspace options

- Choose **Tools** > **Options** from the main menu.
  The workspace options described below appear in a dialog.

To enble or disable a workspace option

- Check or uncheck the corresponding option.

Display Success Messages

These messages appear in a message box and confirm that a command has been performed successfully.

Example: "Save was successful".

Display Delete Messages

These messages appear in a message box and warn you that you are about to do something that you might later regret.

Example: "Do you wish to delete the selected objects?"

Display Replace Messages

These messages appear in a message box when an object is intended to be copied over an already existing one.

Example: "Are you sure you want to replace Program ’SAMPLE’ in library LIB ?"

Exit Prompt

This prompt appears in a message box and warns you that you are about to leave an editor or Natural altogether.

Example: "This will end your Natural session".

Show Full Cat All Result List

If this option is checked, the result list will display all objects processed during Cat All, regardless if the "Cat an Object" returns an error or not.

Perform Automatic Refresh

If this option is checked, the workspace will be automatically refreshed.

If this option is unchecked, the user has to do the Refresh manually (using the "Refresh" command).

Note:
You can also uncheck this option to improve the performance especially when working remotely or working with huge libraries.

Enable Plug-ins

If this option is checked, the Plug-in Manager and all other plug-ins that are marked for automatic activation are activated when Natural Studio is started.

If this option is unchecked, neither the Plug-in Manager nor any other plug-in is activated when Natural Studio is started.

Note:
This option cannot be checked if the profile parameter USEREP is set to ON.

Terminal Emulation

The terminal emulation is used for remote development only. It shows the output of, for example, a report that is executed remotely on a mainframe. For details, e.g. on how to change the font and/or the character set used, see Terminal Emulation in the document Remote Development.

Using Natural Output Window Options

The Natural output window is the window displayed whenever a Natural program writes output to the screen.

The following topics are covered below:

- Output Window Features
- Customizing the Mininimize Icon
- Viewing/Modifying Output Window Profile Settings
- Output Window General Profile
- Output Window Color Profile
- Output Window Font Profile

See also:

- Natural Output Window Information (in the Operations documentation)
  - Output Window Features
  - Output Window Profiling
  - Additional Information on Font

Output Window Features

The Natural output window provides the following features:

- It can be sized and moved.
- If its size is less than the Natural output page, scroll bars appear.
- PF keys defined in a Natural program are converted into push buttons.
  You can use either the push buttons or the keyboard PF keys.
- Clipboard functionality is available.
  Information from Natural output can be cut or copied to the clipboard, and information from the clipboard can be pasted into input fields of the output window.
- Windows created using the Define Window statement are placed into the output window. They are moveable, sizeable and scrollable child windows of the output window.
- The cursor can be positioned using the mouse.
  Double-clicking the left mouse button simulates the ENTER key.
  The system variables *CURSOR, *CURS-COL and *CURS-LINE will be set to the current mouse position.

Customizing the Mininimize Icon

Whenever the Natural output window is minimized, an icon is displayed at the bottom of the screen.

To customize this icon to your needs

- Produce an iconfile (*.ico).
  The iconfile is selected first in the logon libraries RES subdirectory, then in the RES subdirectory of each STEPLIB and then in the directory assigned to the environment variable NATGUI_BMP.
- To use the icon file, issue the statement: `SET CONTROL 'I=`*`iconfile`*`.ICO'`

Viewing/Modifying Output Window Profile Settings

To display the output window options

- From the **Tools** menu, choose **Options...** > **Output Window**.
  The "Natural Output Window Profile" window appears with the profile options explained below.

To modify the output window profile settings

- Check or uncheck the corresponding option.

Output Window General Profile

- Activate report page buffer
- Display input fields with frame
- Display PF-key buttons with number
- Display more prompt
- Disable ESC key
- Disable help menu
- Fixed fonts only

In addition to these general profile options, the push-buttons **Colors** and **Fonts** are available.

Activate report page buffer

Activates buffering, which accommodates approximately 250 lines of Natural output. Input empties the report page buffer.

Display input fields with frame

Displays all input fields with a border (frame).

Display PF-key buttons with number

If selected, the **PF-key** button contains the number of the associated PF key; the name of the PF key is displayed below the "PF-key" button.
If not selected, the **PF-key** button contains the name of the PF key, but the number is not displayed.

Display more prompt

Activates the MORE prompt for output generated by the Natural statements DISPLAY, WRITE or PRINT.

Disable ESC key

Defeats the function the ESC key. When this option is checked, the end user is not able to use ESC to quit the current Natural program.

Disable help menu

If selected, the Natural output window no longer displays a "Help" menu entry. If you do not define any other menu entries, the Natural output window no longer displays a menu bar.

Fixed fonts only

Restricts the "Font selection" dialog to the use of fixed character width fonts only, that is, this option inhibits the use of proportional fonts.

Output Window Color Profile

To invoke the "Color Profiling" dialog box

- Choose the **Colors** button.
  The "Color Profiling" dialog box appears.

To alter the colors to be used for all Natural output text

- Choose the desired color from the list box.

To reset the chosen color scheme

- Click on **Defaults**.
  This will reset your choice to the default color scheme (Normal).

Output Window Font Profile

- Invoking the Front Profile Dialog
- Accepting Changes Made to the Profile Settings

Invoking the Front Profile Dialog

To invoke the "Font profile" dialog box

- Choose the **Fonts** button.
  The "Font profile" dialog box appears.

To select the font to be used for all Natural output text in the output window

- Choose the desired font name, style, and size in the respective list box.

Note:
If you want to use proportional fonts, make sure that the option "Fixed fonts only" has not been marked.

Accepting Changes made to the Profile Settings

If you want to accept the changes you have made to your output window profile setting, select either OK if you want the changes to be valid for the duration of the Natural session, or "Save" if you want them to be valid also for future Natural sessions.

Using Session Parameters

When you start Natural, the Natural parameter file is read to determine the settings of several parameters which customize the Natural installation to your specific environment and requirements. Some of these parameters can be updated after the Natural session is up and running.

Note:
These modifications are only valid for the current session. When you exit Natural, these settings are discarded and the settings in the Natural parameter module are used for the next Natural session.

This section describes parameters that can be dynamically updated, that is, updated during a running session. For a full description of all available session parameters, follow the corresponding parameter links or see the Session Parameters overview page in the Natural Reference documentation.

The following topics are covered below:

- Setting Session Parameters
- Session Parameters Grouped by Function
  - Report Parameters
  - Limit Settings
  - Character Assignments
  - Compiler Options
  - Field Appearance
  - Error Handling
  - Data Representation

Setting Session Parameters

To set global parameters dynamically

- From the **Tools** menu, select **Session Parameters**.
  (To set a parameter to ON check the corresponding box.)
  Or enter the GLOBALS command in the command line.
  Or use the SET GLOBALS statement in a Natural program.
  Or set the session parameter(s) at the statement and/or element level with certain Natural statements (FORMAT, DISPLAY, INPUT, REINPUT, WRITE, PRINT).

Session Parameters Grouped by Function

- Report Parameters
- Limit Settings
- Character Assignments
- Compiler Options
- Field Appearance
- Error Handling
- Data Representation

Report Parameters

Parameter	Function
EJ	Page eject
LS	Line size
PS	Page size
SF	Spacing factor
IM	Default terminal mode

Limit Settings

Parameter	Function
LE	Limit error processing
LT	Processing loop limit

Character Assignments

Parameter	Function
CF	Terminal command character
DC	Decimal-point notation character
IA	Input ASSIGN character
ID	INPUT statement delimiter character

Note:
All character assignments must be mutually exclusive.

Compiler Options

Parameter	Function
DU	Memory dump generation
FS	Length/format specification
SM	Structured mode
SYMGEN	Generate symbol tables

Field Appearance

Parameter	Function
ZP	Zero printing
PM	Print mode
ML	Msg line position
FCDP	Filler characters proteced
OPF	Overwriting protected

Error Handling

Parameter	Function
SA	Terminal alarm feature
ZD	Zero division
REINP	Automatic REINPUT

Data Representation

Parameter	Function
DFOUT	Date format output
DFSTACK	Date format stack
DFTITLE	Date format report title

Accessing Tools

The following topics are covered below:

- Invoking Development Tools
- Development Tools Available

Invoking Development Tools

From the Natural environment, you can access applications that support the software development process.

To access the development tools

- From the menu bar, choose **Tools** > **Development Tools**.
  A list of tools available is displayed.

Development Tools Available

Provided that the corresponding options were chosen during the installation procedure, the following software engineering tools are available from the **Tools** entry of the menu bar:

Application Shell

Invokes Natural Frame Gallery administration. For more information, see the Natural Application Shell documentation.

Component Browser

A tool for browsing existing ActiveX controls. For more information, see the Component Browser.

Debugger

Start the Natural Debugger to find errors on source code level in a Natural application. For more information, see the Debugger documentation.

Error Messages

The Natural utility SYSERR is used for the creation and maintenance of error messages. These error messages can then be used in a Natural application.

- You can write your own application-specific messages.
- You can also modify existing Natural system messages; however, this is not recommended, because these modifications will be lost when a new release of Natural is installed.

For information about creating an error text file, see the Natural SYSERR Utility documentation.

Frame Gallery

Generates an application frame in the Natural Frame Gallery. For more information, see the Natural Frame Gallery documentation.

Natural Reporter

A simple, but powerful tool for generating virtually any type of report directly from a Natural program. For more information, see the Natural Reporter documentation from the HTML Online Help.

Object Handler

The Object Handler is designed to process Natural and non-Natural objects for distribution in Natural environments. This is done by unloading the objects in the source environment to work files and loading them from work files in the target environment. For more information, see the SYSOBJH Utility.

Unlock Objects

Used for remote development only. See Object Locking in the section Remote Development.

User Exits

Displays a list of the existing user exits.

Arranging Your Natural Environment

The following topics are covered below:

- Displaying Your Natural Version
- Arranging Your Natural Environment

Displaying the Natural Version of Your Environment

If you have different versions of Natural for Windows installed on your desktop, it is helpful to get information about the product version of the Natural environment you are currently using.

To display the version of Natural you are currently running

- From the **Help** menu, choose **About Natural Studio**.
  The version number and copyright year for the Natural version is displayed.

Working with Dockable/Floatable and MDI Windows

There are two ways to change the position, location or visibility of items in the Natural environment depending on which type of windows is currently being used:

- Dockable and Floatable Windows
- MDI Windows

Dockable and Floatable Windows

The following windows can be made dockable or floatable:

- Command Line
- All Toolbars and Menus
- Library Workspace
- Application Workspace
- Results Window

For details on these windows, follow the links or refer to the corresponding descriptions in the Natural Studio documentation.

To switch between a dockable and floatable window

1. Position the left mouse on the double bar of the window, keep the button pressed and drag the window to a different position. During this process the window is changing to floatable. When the mouse button is released, the windows docks to the new border.
2. Double click on the double bar toggles the status from dockable to floatable and vice versa.

To hide a dockable or floatable window

- To hide a dockable or floatable window, click on the cross in the upper right corner of the window.
  Or use the **View** menu of the main menu bar.
  Or use the environment context menu.

To show a dockable or floatable window

- Use the main menu bar’s **View** menu.
  Or use the environment context menu.

MDI Windows

The following windows used in the Natural environment follow the MDI concept:

- List Views
- Editors

MDI windows can be created from the library workspace and will be maintained with the **Window** menu.

The **Window** menu allows you to arrange or access windows or icons on your desktop. The following items are on the menu:

Menu Item	Function
Next (CTRL+TAB)	Activates the next inactive MDI window.
Previous (SHIFT+CTRL+TAB)	Activates the previous inactive MDI window.
Close all	Close all MDI windows.
Cascade	Arranges the windows as overlapping tiles.
Tile horizontally	Arranges the windows as horizontal, non overlapping tiles.
Tile vertically	Arranges the windows as vertical, non overlapping tiles.
Arrange icons	Arrange the icons of minimized windows.

In addition, a list of active MDI windows which can be selected for activation.

The windows menu also contains a list of all open MDI windows. The currently active window is marked with a checkbox.

Using Online Help

Online help employs all of the information search and retrieval methods available in typical Windows applications.

- In most cases, help is context sensitive, leading you directly to the information you seek.
- It is also possible, however, to search by topic or index entry.

For more information about using Windows online help, see the Microsoft Windows documentation.

The following topics are covered below:

- Starting Online Help
- Displaying the System Command List
- Displaying Error Message Texts

Starting Online Help

To access the online help overview page

1. Press **F1**.
2. Choose **Contents** from the **Help** menu.

To access the context-sensitive online help

- Choose the **Help** command button you will find in some dialog boxes.

Displaying the System Command List

To display an overview of all Natural system commands available

1. From the **Help** menu, choose **System Commands**.
   The System Command List is displayed
   (described in the Natural Reference documentation).
2. Choose the command from the selection box.

Displaying Error Message Texts

To display a message text for a specific error number

1. From the **Help** menu, choose **Natural Errors**.
   The "Help On Errors" dialog box appears.
2. In the "Error Type" group frame, select "System" for Natural system error messages or "User" for user-defined error messages.
3. In the "Error Number" text box, enter the error number.
4. Choose **OK**.
   Long and short explanations for the error message are displayed.

Natural Studio - Introduction

The following topics are covered:

- Natural Studio - Features and Components
- Library Workspace
- Multiple Document Interface Area
- Toolbars and Menus
- Results Window
- Results Interface
- Command Line
- Status Bar
- Context Menus
- Accelerators
- Commands
- Manual Refresh
- Customizing

See also:

Natural Studio - Fundamentals

Natural Studio - Features and Components

- Natural Studio - Features
- Natural Studio - Main Components

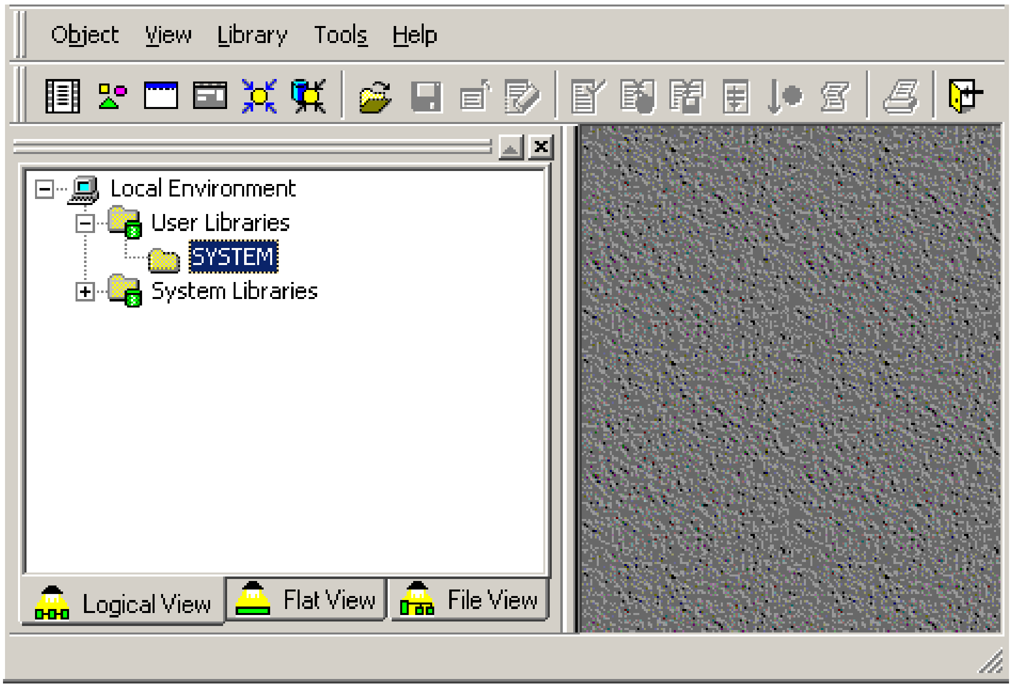
Natural Studio - Features

Natural Studio can be used to create mission-critical enterprise applications quickly and easily. These can be traditional client-server, component-based or web-enabled applications.

Natural applications are portable, scalable and interoperable across multiple computing platforms ranging from the mainframe to the desktop.

The development environment consists of the following main components:

- the Natural programming language and runtime environment, used to access and modify database contents, generate reports, build applications with a graphical user interface, perform calculations, process tables and so on.
- powerful editors to create screen layouts, data areas, dialogs or programs.
- a report writer for creating attractive reports, combining your data, text, charts, and pictures.
- a Component Browser for simplifying the integration of Active X components.
- a Web Interface.
- an integrated Class Builder for creating classes.
- a Debugger.
- various other utilities helpful in application development and system administration, for example, a tool for maintaining error messages.
- a Plug-in Manager making the Natural Studio user interface extensible by plug-ins. Part of the Natural Studio functionality itself is delivered in the form of plug-ins, for example, the XRef GUI Client which is provided in a Natural Single Point of Development (SPoD) environment.
- additional features, such as an application workspace, a terminal emulation window and an XRef GUI Client, are available to enable Natural Studio to be used as a client for remote development in a Single Point of Development (SPoD) scenario.



When the Natural desktop appears for the first time, the library workspace will be displayed on the left side of the development environment with the current logon library selected. When Natural is restarted, the toolbars and windows are placed at the same positions as when Natural was left the last time.

Natural objects are structured in libraries: the default library will be SYSTEM unless a different one is specified by the administrator.

The default library can be altered by setting the Natural profile parameter INIT-LIB in the Natural configuration utility.

For more information, see the Natural Operations documentation.

Natural Studio provides various modern techniques for fast construction of Natural applications. These are:

- Context menus for almost all situations.
- Dockable windows and toolbars with tooltips.
- Full drag&drop / cut&paste support for the file operations copy and move.
- In-place editing for renaming objects or creating new objects.
- Customization of the environment.

Natural Studio - Main Components

Natural Studio consists of the following GUI components:

- Library Workspace
- Application Workspace
- Multiple Document Interface Area
- Toolbars and Menus
- Results Window
- Results Interface
- Command Line
- Status Bar

Library Workspace

The Library Workspace is used to administer all Natural system files in the current environment in a hierarchical manner as a tree view. In Natural terms, a system file is a collection of Natural Libraries and a Natural library is a collection of Natural Objects and Shared Resources.

The workspace is structured and can be displayed in three different views, the logical, the flat and the file view. In addition, the Library Workspace is a dockable window which can be placed at a different position or also can be made floatable or invisible.

From the Library Workspace, any node of the tree can be opened or new objects can be created into the Multiple Document Interface Area.

The following topics are covered below:

- Switching the Library/Application Workspace On or Off
- Local Environment
- Remote Environment
- Logical View
- Flat View
- File View

See also: Application Workspace

Switching the Library/Application Workspace On or Off

To switch the Library/Application Workspace on or off

- Press **ALT - 1**.
  Or, from the menu bar, choose **View** and check **Library Workspace** and/or **Application Workspace**.
  Or open the context menu on any toolbar, in the Multiple Document Interface Area or in the Natural frame window
  and check **Library Workspace** and/or **Application Workspace**..

Local Environment

The local environment is used to develop and run Natural applications for the local Windows environment.

It consists of the currently active user libraries (also known as the system file FUSER) and the currently active system libraries (also known as the system file FNAT). All Natural commands (CAT, STOW, EXECUTE, etc.) and all object operations (Move, Delete, etc.) are supported. See System Commands or Object Operations for further information.

Remote Environment

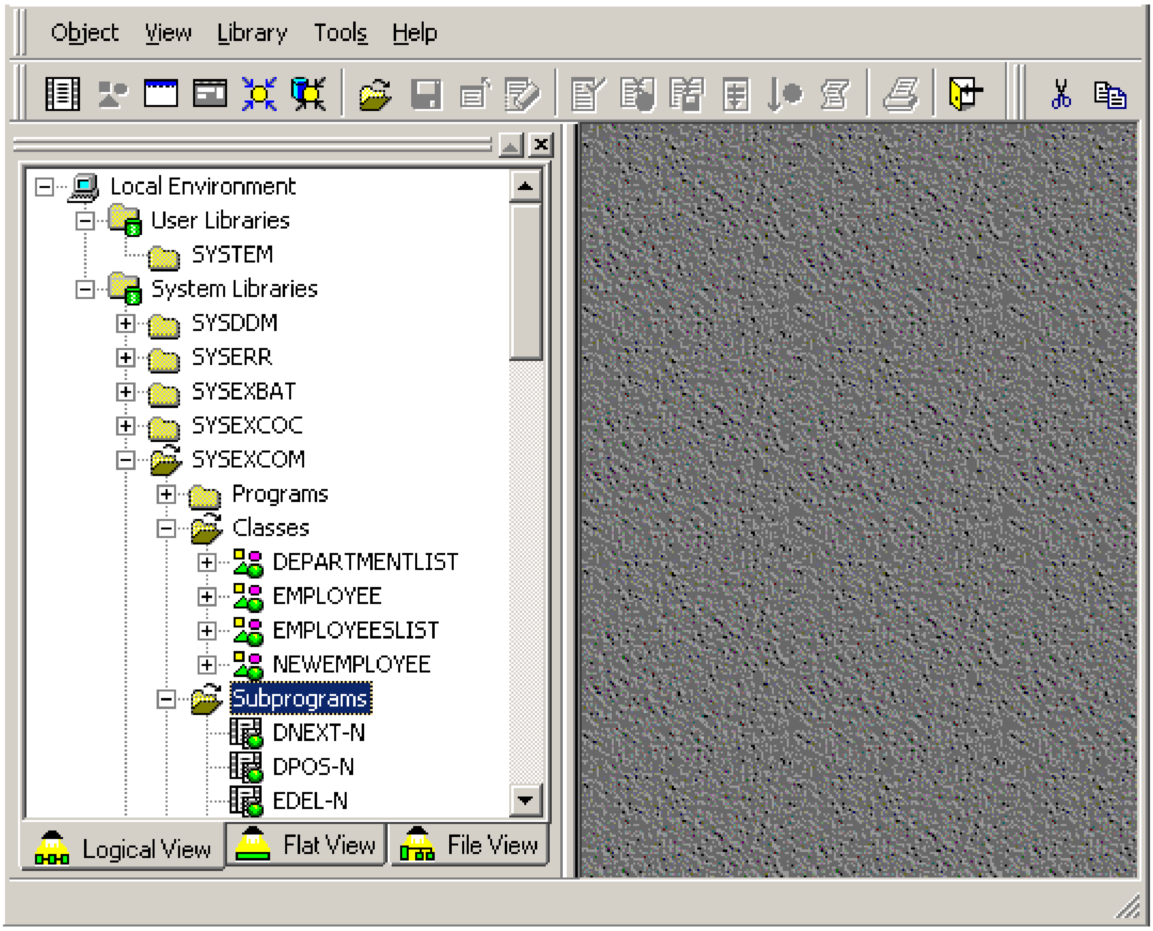
With the introduction of Natural Single Point of Development (SPoD), you can map to a mainframe environment in order to develop applications remotely with the benefits of the powerful graphical user interface. For more information, refer to the Natural SPoD documentation.

Logical View

The logical view displays the objects of libraries in a structured manner. For any Object Types available, a corresponding group node is displayed. For objects of type subroutine, class and DDM, the long name will be displayed. This is in contrast to the file names of the remaining object types such as programs or subprograms.

In addition, it is possible to create new or maintain existing classes using the Class Builder. For information on the visual representation of objects, refer to Object Visualisation.

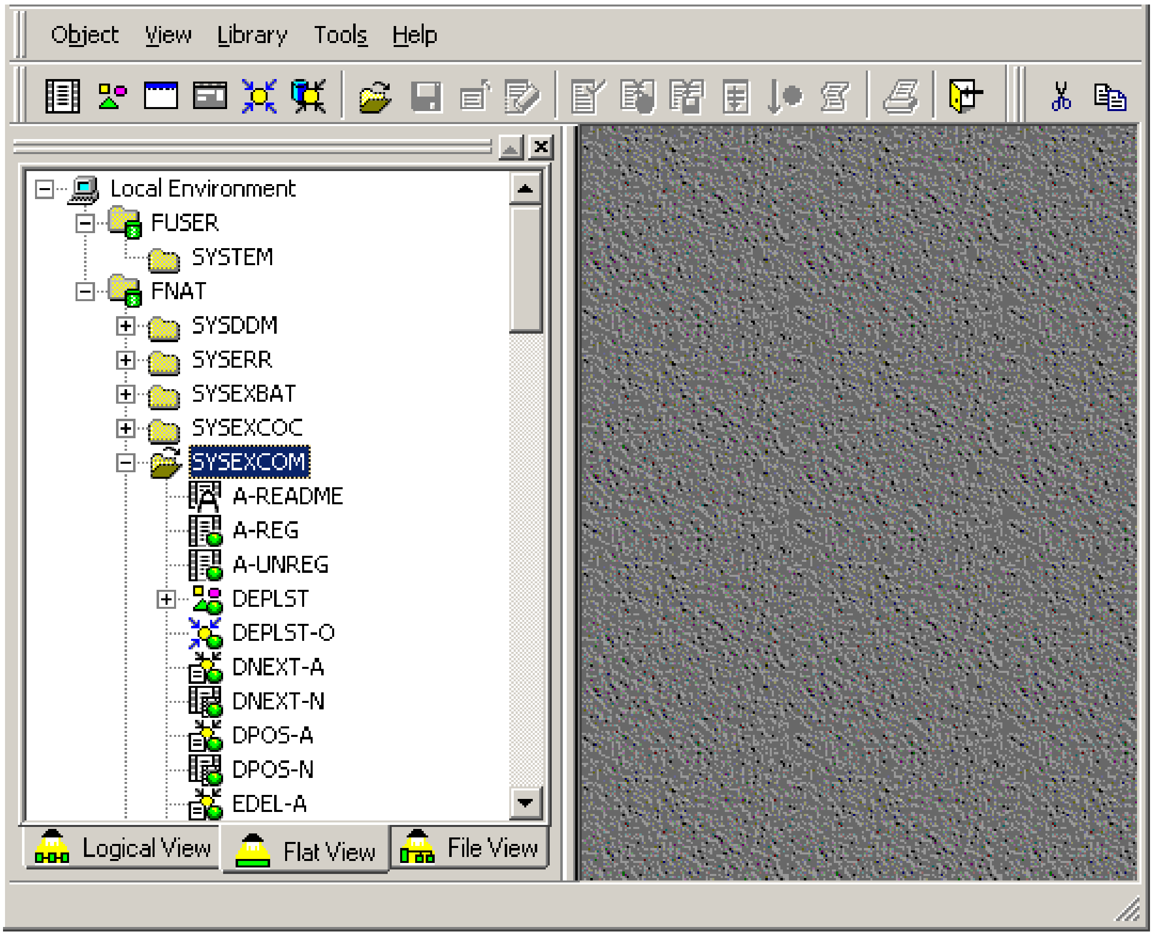
For more information on object types used in Natural, refer to Object Types.



Flat View

The flat view displays the objects of libraries without any grouping.

Note:
To determine what files are represented by an object, also refer to Object Visualisation.



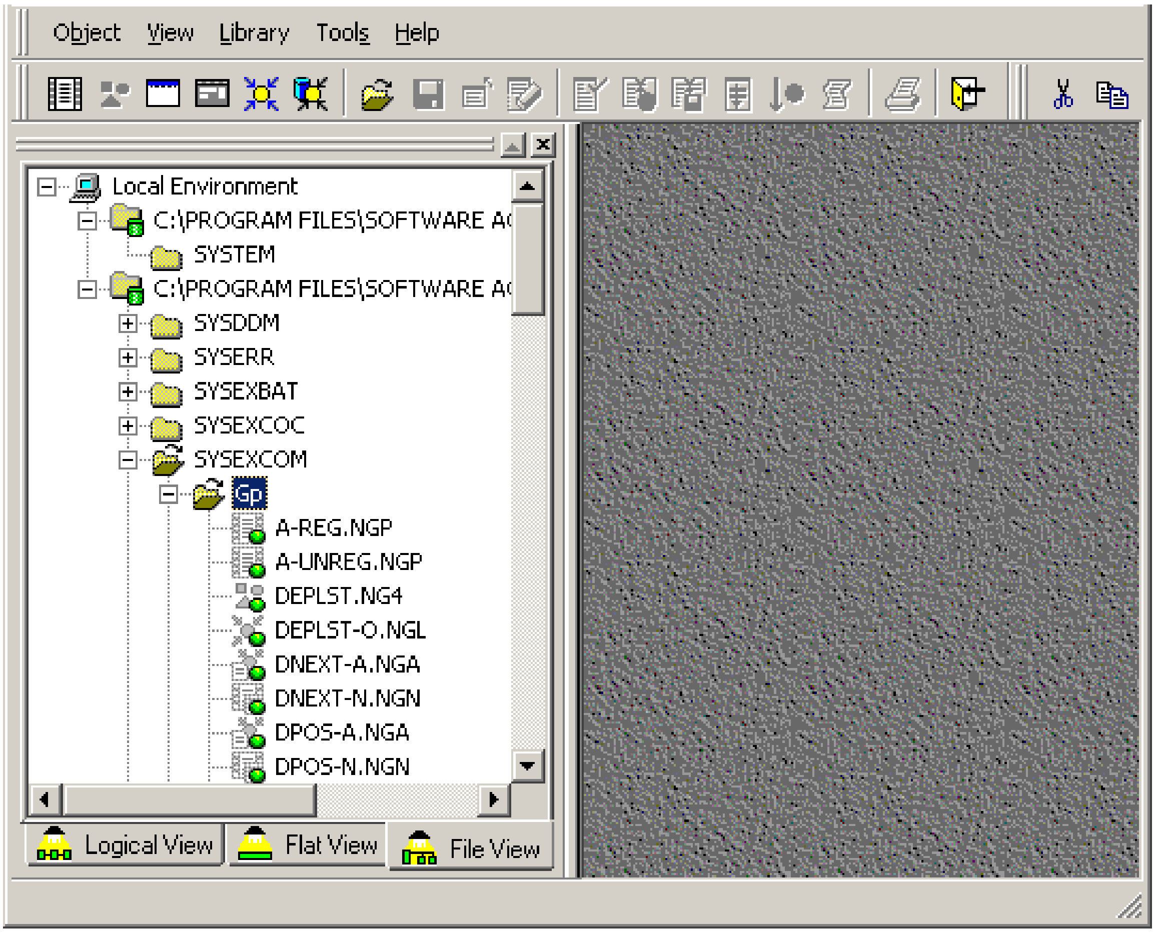
File View

The file view represents the structure of the Natural environment as available on the file system. A Natural system file is displayed as the path to the "Src" and "Gp" subdirectory, for example:

```
D:\SAG\NAT\V511\FNAT
```

The "Src" directory contains all Natural sources and the "Gp" subdirectory contains all Natural generated programs (Natural executables). The Natural objects are displayed with the corresponding file extensions (for example, ".NGP" for Natural Generated Program or ".NSP" for Natural Source Program).

Note:
To determine what files are represented by an object, also refer to Object Visualisation.



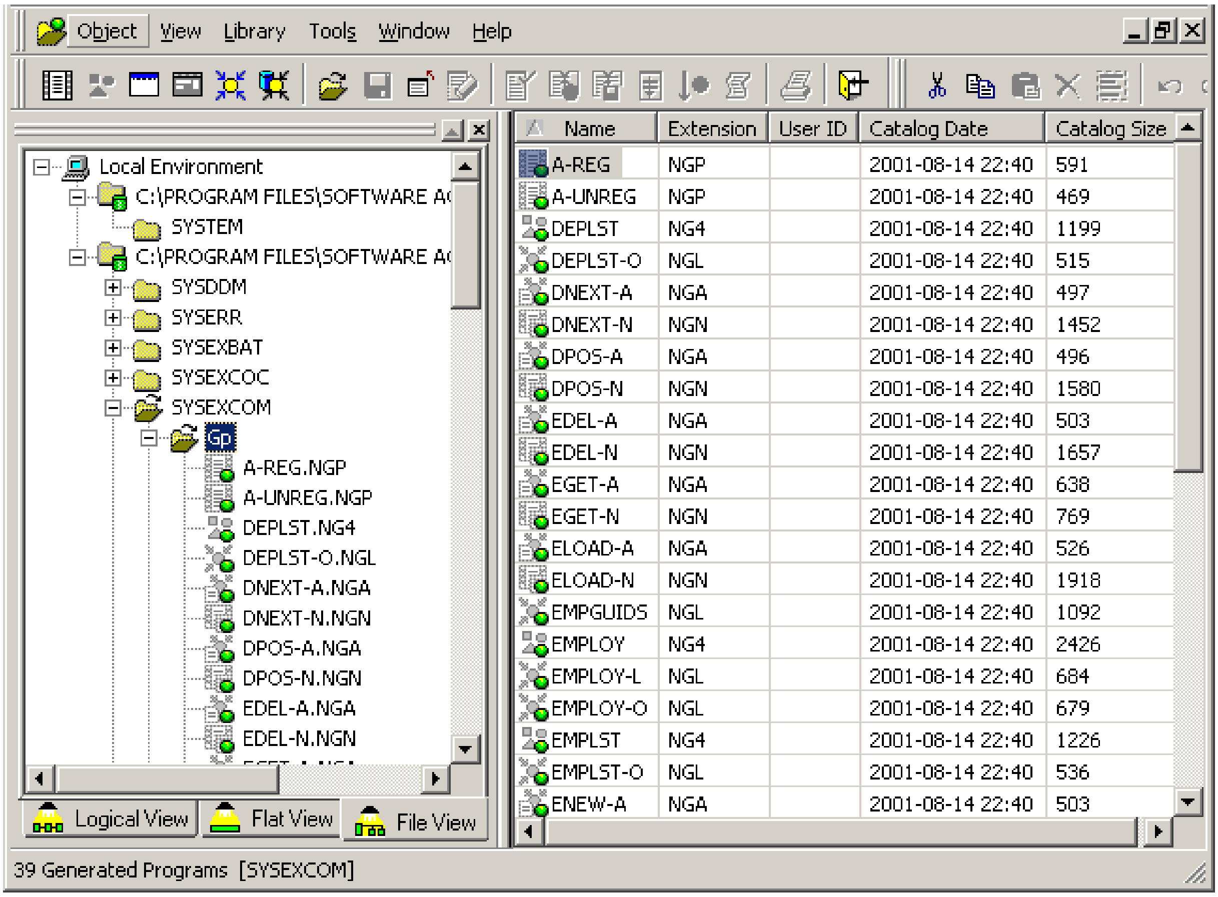
Multiple Document Interface Area

From the Library Workspace, any node of the tree can be opened or new objects can be created into the Multiple Document Interface (MDI) area. The possible objects are editor objects and list view objects.

- List View
- Editors Available in MDI Area
- MDI Window Navigation Accelerators

List View

For every node in the library workspace (except the object nodes which will be processed in the corresponding object editor), a List View can be opened in order to display more detailed information on the selected node.



Name	Extension	User ID	Catalog Date	Catalog Size
A-REG	NGP		2001-08-14 22:40	591
A-UNREG	NGP		2001-08-14 22:40	469
DEPLST	NG4		2001-08-14 22:40	1199
DEPLST-O	NGL		2001-08-14 22:40	515
DNEXT-A	NGA		2001-08-14 22:40	497
DNEXT-N	NGN		2001-08-14 22:40	1452
DPOS-A	NGA		2001-08-14 22:40	496
DPOS-N	NGN		2001-08-14 22:40	1580
EDEL-A	NGA		2001-08-14 22:40	503
EDEL-N	NGN		2001-08-14 22:40	1657
EGET-A	NGA		2001-08-14 22:40	638
EGET-N	NGN		2001-08-14 22:40	769
ELOAD-A	NGA		2001-08-14 22:40	526
ELOAD-N	NGN		2001-08-14 22:40	1918
EMPGUIDS	NGL		2001-08-14 22:40	1092
EMPLOY	NG4		2001-08-14 22:40	2426
EMPLOY-L	NGL		2001-08-14 22:40	684
EMPLOY-O	NGL		2001-08-14 22:40	679
EMPLST	NG4		2001-08-14 22:40	1226
EMPLST-O	NGL		2001-08-14 22:40	536
ENEW-A	NGA		2001-08-14 22:40	503

39 Generated Programs [SYSEXCOM]

Opening a List View

To open a List View

1. Select the node in the library workspace or in an already open List View.
2. Open the context menu and select the **Open** command.
   Or open the **Object** menu in the menu bar and select the **Open** command.
   Or use the accelerator key **CTRL+O**.
   Or press **ENTER.**

List View Operations

A List View supports the following operations:

- **Repositioning the columns using header drag and drop.**
  D rag a header of a specific column and drop it onto a different header position. For example, drag the "Extension" column of the "Generated Programs" List View and drag it to the beginning in order to position this column as the first one;
- **Resizing the column width.**
- **Sorting columns in ascending and descending order; sorting a single column or sorting multiple columns.**
  It is, for example, possible to click on the "Extension" column header to sort the column in ascending order and then hold the **CTRL** key down and click on the "Name" column twice to sort it in descending order (inside the sort sequence of the column first sorted);
- **Multiple selection for mass operations**
  (in contrast to the library workspace, where only one node can be selected). To select multiple nodes, press the **SHIFT** key (to select items consecutively) or press the **CTRL** key (to select individual items).
- **Changing the layout of the date and time output using the Natural profile parameter DTFORM.**
  For example, it is possible to change the representation from English to German.

Editors Available in MDI Area

The following Editors can be opened in the Multiple Document Interface Area:

- Program Editor
- Map Editor
- Data Area Editor
- DDM Editor
- Dialog Editor

MDI Window Navigation Accelerators

The following MDI-specific accelerator keys can be used to navigate within the Natural environment.

- **CTRL-D** to toggle between the command line and the active MDI window (editor or list view).
- **CTRL-W** to toggle between the active MDI window and the library workspace.
- **CTRL-T** to toggle between the active MDI window and the result view.

For navigation inside the open MDI windows (editors and list views), refer to MDI Windows.

Toolbars and Menus

- Introduction
- Hiding a Toolbar
- Positioning a Toolbar

For information on customizing a toolbar, refer to the following topics concerning the "Customize" dialog:

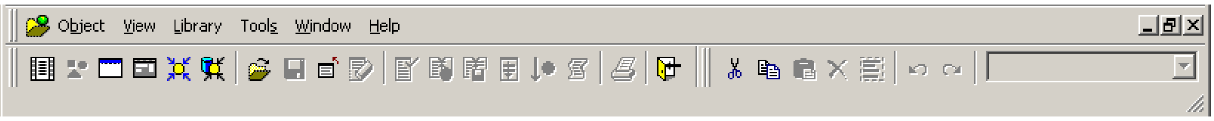
- Creating a Toolbar
- Editing a Toolbar
- Adding User-Defined Commands to a Toolbar
- Removing Commands from a Toolbar
- Rearranging Commands in the Toolbar
- Selecting a Toolbar
- Removing a Toolbar

Introduction

The menu bar and toolbar offer the following features:

- **All major commands can be issued using the menu bar**.
  This is the bar displayed at the top of the Natural development environment. Like all available toolbars, the menu bar is dockable and floatable and therefore can be placed in a different position.
- **The menus offered in the menu bar vary depending on the active window.**
  The selection of commands from the menu bar is performed just like all graphic-based applications. Another way of issuing commands is using a context menu by clicking the right mouse button on the corresponding node.
- **Toolbar buttons provide an alternative to menu commands.**
  Buttons are avilable for the most commonly used functions such as saving, checking, running, and stowing objects. For example, to open a library, you can click the "Open selected object" toolbar button instead of opening a menu and choosing a command.
- **Each toolbar button is self-documenting.**
  Just place the mouse pointer on the toolbar button, and read the text in the status line. Additionally a tool tip is provided for the command (when the corresponding option is switched on in the "Customize" dialog).
- **The toolbar is context sensitive**.
  It displays only the buttons that apply to the active window. In general, each editor has its own toolbar.

The first time Natural is started, the default toolbars that are provided for each window are displayed. However, new buttons can be added, existing ones can be removed or rearranged and even new toolbars can be created using the "Customize" dialog.



Hiding a Toolbar

You can hide a toolbar when you are not using it.

To hide your toolbar

1. Click with the right mouse button on any toolbar.
2. From the resulting context menu, select the toolbar to be hidden.
   (If the check mark is not visible, the toolbar is hidden.)

Positioning a Toolbar

The toolbar is located by default at the top of your desktop just below the menu bar, but you can dock it by drag and drop along any other edge of your desktop.

To position your toolbar on the desktop

1. Select the toolbar to be repositioned at its far left.
2. Press and hold the left mouse button.
3. Using drag and drop, place the toolbar into position.

Results Window

- Switching the Results Window On and Off
- Using the Results Window

See also Results Interface.

Switching the Results Window On and Off

To switch the result window on or off

- Press **ALT - 5**.
  Or, in the menu bar's **View** menu, check **Results**.
  Or, in the environment context menu on any toolbar, on the Natural document interface area on the Natural frame window, check **Results**.

Using the Results Window

Natural Studio provides a window displaying the results of the commands CAT ALL and FIND OBJECTS as well as the user defined tabs created by the Results Interface.

- As with the Library Workspace, this window is also dockable and floatable.
- The results can be switched by pressing the corresponding Tab key.
- For further processing, the same commands (except file operations like Copy or Delete) as in the library workspace can be used (for example, execution of a found object).
- As the window is a List View, the same operations for organizing the list as described in the List View are available.

Results Interface

The following topics are covered:

- Purpose of the Results Interface
- Results Window Control Bar Access
- Tab Handling
- Image Handling
- Context-Menu Handling
- Command Handling
- Column Handling
- Row Handling
- Data Handling
- Selection Handling

See also Results Window.

Purpose of the Results Interface

The Results Interface enables programmers to display data within the results window. The results of CAT ALL and FIND OBJECTS are not affected by the Results Interface.

The design and the usage of a tab can be determined via user exits. In general, a detailed view with columns and lines is used.

Context menus can be created for each entry, so that after the user-defined tab is shown it can be used for further processing.

This processing has to be defined within two programs.

1. in an update Command Handler before a context menu is shown;
2. in a Command Handler if an item is selected.

The user exits for the Results Interface are **USR5001N - USR5016N** and can be found in the library SYSEXT.

An example of the various functions is available in **USR5001P** with the command handler in **USR5001A** and **USR5001B**.

Note:
Modifications of the pre-defined tabs (e.g. **Find Objects** and **Cat All**) are not possible with this interface.
The Results Window and the Results Interface can be accessed only from Natural Studio.

Results Window Control Bar Access

In this section, the Results Window Control Bar can be accessed.

User Exit	Functionality
USR5001N	Turns Results Window on/off. Checks visibility of the Results Window.

Tab Handling

In this section, the general layout of a tab can be defined.
A tab can contain all or one of the following:

- Check Box
- Full Row selection
- Single Row selection
- Images

A tab can be defined with the following attributes:

- Layout of the view (Large / small icons, list or details view)
- Several usages (Check boxes, images, grid lines, full or single row selection, view change)
- Layout of the tab label (Text, bitmap or icon)

User Exit	Functionality
USR5004N	Add, replace and delete a tab.
USR5005N	Set and get active tab. Set tab active and set the focus on this tab

Image Handling

In this section, bitmaps (*.bmp) and icons (*.ico) can be specified for a previously defined tab.

User Exit	Functionality
USR5002N	Add and delete bitmaps and icons for a specified tab.

Context-Menu Handling

In this section, user-defined context-menus can be specified.

User Exit	Functionality
USR5003N	Add, remove and delete context-menus of a tab.
USR5007N	Set and get checked/enabled state of context-menu items.

The hierachy of the context-menu must be defined manually.

The following array components can be defined:

Array Component	Value	Description
Type	1 to 4	1 - Context-Menu Handling 2 - Separator line 3 - Begin of submenu 4 - End of submenu
Command ID	1 to 255	Free selectable number to identify a certain item in a context menu (used within the command handler).
Label	alpha-nummeric text	Text for the context-menu items of type 1 and 3. A text for the status bar can be separated with H’0’A.
Image	Handle of image	Handle of a previously defined image (bitmap or icon). The image will be placed before the text of the context-menu item.

Command Handling

A program can be assigned as an update command handler or as a command handler.
User defined data can be saved / restored in the internal work area of the command handlers.

For example: handles of tabs.

User Exit	Functionality
USR5006N	Define update command handler and command handler.
USR5016N	Set and get data for the command handler work area.

Column Handling

In this section, the general layout of a column can be defined.
A column can contain all or one of the following:

- Title
- Width
- Data position
- Column sort

In addition, the default width and specified width of the column can be set up individually.

User Exit	Functionality
USR5008N	Add, insert and delete columns of a tab.
USR5010N	Set and get default column width and width for specified columns.
USR5009N	Count number of columns.

Row Handling

In this section, the rows with images and context menus can be defined.

User Exit	Functionality
USR5011N	Add, insert and delete rows of a tab.
USR5009N	Count number of rows.

Data Handling

In this section, user defined data can be written into defined columns/rows.
If check boxes have been defined for a tab, then they can be activated/deactivated for every row.

User Exit	Functionality
USR5012N	Set and get data into a tab.
USR5013N	Set and get checked state of a row.

Selection Handling

In this section, rows can be individually selected.

User Exit	Functionality
USR5014N	● Set and get selected rows. ● Count amount of selected rows. ● Reset row selection.
USR5015N	Set and get row of focus.

Command Line

The command line provides another way of using Natural System Commands. The following topics are covered below:

- Using the Command Line
- Display or Hide the Command Line
- Associate an Object with the Command Line

Using the Command Line

When Natural Studio is started for the first time, the command line is switched off, since in most cases a command can be issued using the menu bar, a context menu or a toolbar command, thus making the command line obsolete.

But one can still imagine some scenarios where applying commands with the command line is quite useful. This is especially the case for commands which are not dependent on the context. An example of a context-sensitive command could be the STOW command which always acts on the current selection in the library workspace or an active List View. It is used to set global settings such as `GLOBALS SYMGEN = ON` (generation of symbolic information for any object to be cataloged).

The command line offers the following features:

- **Direct commands**
  For users who are already quite familiar with the Natural system commands, it might be faster to issue commands using the command line rather than selecting the command with a context menu or with a toolbar button.
- **Command history mechanism**
  For faster command execution, the command line uses a command history mechanism in conjunction with reselecting commands already entered. On the right side, the currently active library (the library where the user is currently logged in) is displayed.
- **Dockable and floatable**
  As with the menu bar, the library workspace and the toolbars, the command line can be docked to any other position or can be made floatable.

Display or Hide the Command Line

To switch the command line on or off

- Press **ALT - 3**
  Or, from the menu bar, choose **View** and check **Command Line**.
  Or open the context menu on any toolbar in the Multiple Document Interface Area.

Associate an Object with the Command Line

In some cases, it is not obvious on which selected object the command in the command line was applied since it is possible that both in the Library Workspace and the List View some nodes are selected.

For such a scenario, the following rule applies:

The command is always connected to the node which last had the focus.

Status Bar

- Purpose of the Status Bar
- Switching the Status Bar On or Off
- Status Bar Option

Purpose of the Status Bar

The status bar at the bottom of the development environment displays information about the command currently selected or on the progress of an operation currently being performed. Also, information on the currently active list view is displayed.

Switching the Status Bar On or Off

To switch the status bar on or off

- Press **ALT - 4**.
  Or, from the menu bar, choose **View** and check **Status Bar**.
  Or open the environment context menu on any toolbar, in the Natural document interface area or the Natural frame window.

Status Bar Option

The status line at the bottom of your desktop displays the active command messages and the time and date.

To hide the status bar

1. At any position in the main menu bar, press the right mouse button.
2. Click on the status bar to be hidden.
   (If the check mark is not visible, the status bar is hidden.)

Context Menus

Natural Studio supplies Context Menus for all sorts of objects.

Context Menus can be activated with the right mouse button or with the WINDOWS application key for the following windows/nodes:

- any node in the Library Workspace.
- any node in a List View.
- any List View window when no node is selected.
- the frame window, the multiple document interface area or any toolbar.
- any active window.

Accelerators

Many Natural commands are mapped to corresponding accelerator keys. The following topics are covered below:

- Accelerators Grouped by Categories
    - Generally Available Accelerators
    - Dialog Editor Accelerators
    - Program Editor Accelerators
    - Data Area Editor Accelerators
- Alphabetical Overview of Accelerators

See also:

- Changing Accelerator Key Assignments

Accelerators Grouped by Categories

Generally Available Accelerators

The following table shows the accelerators that are available in Windows Explorer, on a selected node in the library workspace or in an active list view.

Accelerator	Description
ALT-ENTER	Display properties of Natural objects
CTRL-C	Copies.
CTRL-O	Opens a Natural object or a list view.
CTRL-P	Prints an object.
CTRL-V	Pastes.
CTRL-X	Cuts.
DEL	Deletes.
ENTER	Opens or executes Natural object or opens a list view.

The following table shows the accelerators that are available anywhere inside Natural.

Accelerator	Description
ALT-1	Toggles view of library workspace.
ALT-2	Toggles view of application workspace.
ALT-3	Toggles view of command line.
ALT-4	Toggles view of status bar.
ALT-5	Toggles view of result list.
ALT-F4	Exits the Natural session.
CTRL-D	Toggles between the command line and the active MDI window.
CTRL-T	Toggles between the result view and the active MDI window.
CTRL-TAB	Makes next MDI window active.
CTRL-W	Toggles between the library workspace and the active MDI window.
CTRL-SHIFT-TAB	Makes previous MDI window active.

Dialog Editor Accelerators

The following table shows the accelerators that are available in the active dialog editor.

Accelerator	Description
CTRL-A	Selects all.
CTRL-E	Checks an object.
CTRL-F	Finds items.
CTRL-H	Replaces items.
CTRL-P	Prints an object.
CTRL-R	Runs an object.
CTRL-S	Saves an object.
CTRL-V	Pastes.
CTRL-X	Cuts.
CTRL-Y	Performs a redo.
CTRL-Z	Performs an indo.
CTRL-ALT-DOWN	Aligns selected controls down.
CTRL-ALT-E	Opens the dialog event handlers dialog.
CTRL-ALT-G	Opens the global data area dialog.
CTRL-ALT-H	Opens the help organizer dialog.
CTRL-ALT-I	Opens the timers dialog.
CTRL-ALT-L	Opens the local data area dialog.
CTRL-ALT-LEFT	Aligns selected controls left.
CTRL-ALT-M	Opens the menu dialog.
CTRL-ALT-O	Opens the comment dialog.
CTRL-ALT-P	Opens the parameter data area dialog.
CTRL-ALT-Q	Displays control sequence numbers.
CTRL-ALT-RIGHT	Aligns selected controls right.
CTRL-ALT-S	Opens the inline subroutine dialog.
CTRL-ALT-T	Opens the toolbar dialog.
CTRL-ALT-UP	Aligns selected controls up.
CTRL-ALT-X	Opens context menu dialog.
CTRL-SHIFT-D	Deletes line.
CTRL-SHIFT-E	Opens event handlers for selected control.
CTRL-SHIFT-F9	Aligns selected controls centered.
CTRL-SHIFT-K	Delete to end of line.
F3	Repeats "Find items".
F9	Vertical center of selected controls.
SHIFT-F9	Horizontal center of selected controls.

Program Editor Accelerators

The following table shows the accelerators that are available in the active program editor.

Accelerator	Description
CTRL-A	Selects all.
CTRL-E	Checks an object.
CTRL-F	Finds items.
CTRL-G	Goto.
CTRL-H	Replaces items.
CTRL-P	Prints an object.
CTRL-R	Runs an object.
CTRL-S	Saves an object.
CTRL-V	Pastes.
CTRL-X	Cuts.
CTRL-Y	Performs a redo.
CTRL-Z	Performs an indo.
CTRL-SHIFT-D	Deletes line.
CTRL-SHIFT-K	Delete to end of line.
CTRL-SHIFT-R	Starts recording.
CTRL-SHIFT-S	Stops recording.
CTRL-SHIFT-P	Replays recording.
F3	Repeats "Find items".
F6	Toggles between split windows.

Data Area Editor Accelerators

The following table shows the accelerators that are available in the active data area editor.

Accelerator	Description
CTRL-A	Selects all.
CTRL-E	Checks an object.
CTRL-F	Finds items.
CTRL-H	Replaces items.
CTRL-P	Prints an object.
CTRL-R	Runs an object.
CTRL-S	Saves an object.
CTRL-V	Pastes.
CTRL-X	Cuts.
CTRL-Y	Performs a redo.
CTRL-Z	Performs an indo.
CTRL-SHIFT-D	Deletes line.
CTRL-SHIFT-I	Jumps to next level.
CTRL-SHIFT-J	Jumps to previous level.
CTRL-SHIFT-K	Delete to end of line.
F3	Repeats "Find items".

Alphabetical Overview of Accelerators

The following alphabetical list shows the default accelerator assignments inside Natural.

Accelerator	Description	Applicable in
ALT-1	Toggles view of library workspace.	Anywhere inside Natural.
ALT-2	Toggles view of application workspace.	Anywhere inside Natural.
ALT-3	Toggles view of command line.	Anywhere inside Natural.
ALT-4	Toggles view of status bar.	Anywhere inside Natural
ALT-5	Toggles view of result list.	Anywhere inside Natural
ALT-ENTER	Display properties of Natural objects	Selected node in library workspace or active list view.
ALT-F4	Exits the Natural session.	Anywhere inside Natural.
ALT-RIGHT	Horizontal spacing.	Active Dialog Editor
ALT-UP	Vertical spacing.	Active Dialog Editor
CTRL-A	Selects all.	Active editor.
CTRL-C	Copies.	Windows explorer, selected node in library workspace or in active list view.

Accelerator	Description	Applicable in
CTRL-D	Toggles between the command line and the active MDI window.	Anywhere inside Natural.
CTRL-E	Checks an object.	Active editor.
CTRL-F	Finds items.	Active editor.
CTRL-F4	Closes the active MDI window (list view or editor).	Active MDI Window.
CTRL-G	Goto.	Active Program Editor.
CTRL-H	Replaces items.	Active editor.
CTRL-N	Creates a new Program.	Active environment.
CTRL-O	Opens a Natural object or a list view.	Selected Node in library workspace or active list view.
CTRL-P	Prints an object.	Active Editor, selected node in library workspace or active list view.
CTRL-R	Runs an object.	Active editor.
CTRL-S	Saves an object.	Active editor.
CTRL-T	Toggles between the result view and the active MDI window.	Anywhere inside Natural.
CTRL-TAB	Makes next MDI window active.	Anywhere inside Natural.
CTRL-V	Pastes.	Windows explorer, selected node in library workspace or in active list view.
CTRL-W	Toggles between the library workspace and the active MDI window.	Anywhere inside Natural.
CTRL-X	Cuts.	Windows explorer, selected node in library workspace or in active list view.
CTRL-Y	Performs a redo.	Active editor.
CTRL-Z	Performs an indo.	Active editor.
CTRL-ALT-C	Displays source code.	Active Dialog Editor.
CTRL-ALT-DOWN	Aligns selected controls down.	Active Dialog Editor.
CTRL-ALT-E	Opens the dialog event handlers dialog.	Active Dialog Editor.
CTRL-ALT-G	Opens the global data area dialog.	Active Dialog Editor.
CTRL-ALT-H	Opens the help organizer dialog.	Active Dialog Editor.
CTRL-ALT-I	Opens the timers dialog.	Active Dialog Editor.
CTRL-ALT-L	Opens the local data area dialog.	Active Dialog Editor.
CTRL-ALT-LEFT	Aligns selected controls left.	Active Dialog Editor.
CTRL-ALT-M	Opens the menu dialog.	Active Dialog Editor.
CTRL-ALT-O	Opens the comment dialog.	Active Dialog Editor.
CTRL-ALT-P	Opens the parameter data area dialog.	Active Dialog Editor.
CTRL-ALT-Q	Displays control sequence numbers.	Active Dialog Editor.

Accelerator	Description	Applicable in
CTRL-ALT-RIGHT	Aligns selected controls right.	Active Dialog Editor.
CTRL-ALT-S	Opens the inline subroutine dialog.	Active Dialog Editor
CTRL-ALT-T	Opens the toolbar dialog.	Active Dialog Editor.
CTRL-ALT-UP	Aligns selected controls up.	Active Dialog Editor.
CTRL-ALT-X	Opens context menu dialog.	Active Dialog Editor.
CTRL-SHIFT-D	Deletes line.	Active editor.
CTRL-SHIFT-E	Opens event handlers for selected control.	Active Dialog Editor.
CTRL-SHIFT-F9	Aligns selected controls centered.	Active Dialog Editor.
CTRL-SHIFT-R	Starts recording.	Active Program Editor.
CTRL-SHIFT-S	Stops recording.	Active Program Editor.
CTRL-SHIFT-P	Replays recording.	Active Program Editor.
CTRL-SHIFT-I	Jumps to next level.	Active Data Area Editor.
CTRL-SHIFT-J	Jumps to previous level.	Active Data Area Editor.
CTRL-SHIFT-K	Delete to end of line.	Active Editor.
CTRL-SHIFT-TAB	Makes previous MDI window active.	Anywhere inside Natural.
DEL	Deletes.	Selected node in library workspace or in active list view.
ENTER	Opens or executes Natural object or opens a list view.	Selected node in library workspace or active list view.
F3	Repeats "Find items".	Active editor.
F6	Toggles between split windows.	Active Program Editor.
F9	Vertical center of selected controls.	Active Dialog Editor
SHIFT-F9	Horizontal center of selected controls.	Active Dialog Editor

Commands

- Purpose of Natural Commands
- Issuing Natural Commands

See also:

- Command Line
- System Commands

Purpose of Natural Commands

Natural Studio uses various commands

- to customize the current environment,
- to position windows, to launch tools, and
- to activate editors.

Commands can also be used in Natural programs (for example, in conjunction with the Natural stack) or used in the Command Line to perform certain operations like stowing an object.

These System Commands can also be entered at a MORE prompt during the execution of a program. A MORE prompt is displayed at the bottom of an output screen to signal that more output is pending. When a system command is entered in response to a MORE prompt, program execution is interrupted and the system command is executed.

Issuing Natural Commands

Commands can be issued in any of the following ways:

- **Select a menu command from the menu bar.**
  Click on the menu or use the access key **ALT** and the arrow keys to select the command.
- **Select a menu command from a context menu.**
  (activated with the right mouse button or with the Windows application key).
- **Click a tool-bar button.**
  A text description of each toolbar button is displayed as a tool tip and in the status bar when you place the arrow pointer on the button.
- **Use accelerator keys.**
  (e.g **CTRL+O** to open an object). See also Accelerators.
- **Use the command line.**
  System commands can be entered in the Command Line or at the MORE prompt of a running application (e.g. SAVE).

Manual Refresh

- Purpose of a Manual Refresh
- Performing a Manual Refresh

Purpose of a Manual Refresh

Usually when something changes in Natural Studio, an automatic refresh is executed. This happens for example, when objects are deleted or renamed or new objects are created, or when the user has switched off the automatic refresh function (Tools > Options > Perform automatic refresh).

There are, however, some situations where an automatic refresh does not take place, since Natural is not aware of the modification.

As an example, imagine that two Natural processes are currently active both of which are working on the same system file. When one Natural is applying a change to the system file (for example, creating a new object), the second Natural is not aware of this modification. For such cases, the manual refresh can be used.

Performing a Manual Refresh

The manual refresh can only be performed on a selected system file node in the Library Workspace or on a selected system file node of a Local Environment or Remote Environment list view.

To perform a manual refresh

1. Select the system file node to be refreshed.
2. Open the context menu or the **View** menu in the menu bar and check **Refresh**
   or simply press **F5** to apply the operation.

Customizing

This overview page summarizes the settings you can make on the sheets of the "Customize..." dialog to adapt the user interface of Natural Studio to your requirements. The following topics are covered:

- Commands
  - Editing a Toolbar
  - Adding User-Defined Commands to a Toolbar
  - Removing Commands from a Toolbar
  - Rearranging Commands in the Toolbar
- Toolbars
  - Invoking the Toolbars Dialog Box
  - Creating a Toolbar
  - Selecting a Toolbar
  - Removing a Toolbar
- Keyboards
  - Changing Accelerator Key Assignments
  - Removing a Key Assignment
  - Resetting Your Personal Key Assignments
- User Commands
  - Invoking the User Commands Dialog
  - Defining a User Command
  - Adding a User Command to a Toolbar

Commands

The following topics are covered:

- Editing a Toolbar
- Adding User-Defined Commands to a Toolbar
- Removing Commands from a Toolbar
- Rearranging Commands in the Toolbar

See also:

- Toolbars and Menus
- Command Line
- System Commands

Editing a Toolbar

To edit a toolbar

1. From the **Tools** menu, select "Customize..."
2. From the "Commands" dialog box, select a category.
3. Select a command and, using drag and drop, place the icons needed into the new toolbar at the top of the screen.
4. Close the dialog box.

Adding User-Defined Commands to a Toolbar

To define and add user-specific commands to a toolbar, proceed as described under User Commands.

Removing Commands from a Toolbar

1. From the **Tools** menu, open the "Customize..." dialog box.
2. Select the command icon to be removed from the toolbar.
3. Click the right mouse button.
   From the resulting context menu, choose **Delete** to remove the command icon.

Rearranging Commands in the Toolbar

1. From the **Tools** menu, open the "Customize..." dialog box.
2. Choose the command icon in the toolbar to be rearranged.
3. Using drag and drop, place the icon into the selected toolbar.

Toolbars

The first time Natural is started, the default toolbars that are provided for each window are displayed. However, new buttons can be added, existing ones can be removed or rearranged and even new toolbars can be created using the "Toolbars" sheet of the "Customize" dialog.

- Invoking the Toolbars Dialog Box
- Creating a Toolbar
- Selecting a Toolbar
- Removing a Toolbar

See also:

- Editing a Toolbar
- Adding User-Defined Commands to a Toolbar
- Removing Commands from a Toolbar
- Rearranging Commands in the Toolbar

Invoking the Toolbars Dialog Box

To invoke the dialog

1. From the **Tools** menu, select "Customize".
2. Click on the "Toolbars" tab.
   The "Toolbars" dialog box appears.

Creating a Toolbar

To create a new toolbar

1. In the "Toolbars" dialog box, click **New** to create a new toolbar.
2. In the "Toolbar Name" field, enter the name of the new toolbar.
3. Close the dialog box.
   The new toolbar will be added to the existing toolbars. Proceed as described in Editing a Toolbar.

Selecting a Toolbar

To select a toolbar

1. From the drop-down combo box in the "Toolbars" dialog box, choose the toolbar to be selected.
2. Choose **Close**.
   The dialog box closes and the selected toolbar is displayed in the Natural desktop.

Removing a Toolbar

To remove a toolbar

1. From the " toolbars" list box in the "Toolbars" dialog box, click on the user defined toolbar to be removed.
2. Click on the **Delete** button and the selected toolbar will be removed from the Natural desktop.

Keyboards

The first time Natural is started, the default keyboard assignments that are provided for each dialog or window are valid. New assignments can be made using the "Keyboards" sheet of the "Customize..." dialog.

The following topics are covered below:

- Changing Accelerator Key Assignments
- Removing a Key Assignment
- Resetting Your Personal Key Assignments

See also:

- Accelerators

Changing Accelerator Key Assignments

Using the "Customize" dialog, you can change or remove any accelerator key assignments.

To change the accelerator key assignments

1. From the **Tools** menu, select **Customize** and click on the "Keyboard" tab.
2. From the "Category" list box, choose the category the command belongs to
   or choose "All Commands".
3. Select the command whose accelerator key assignment is to be changed.
   The current (default) assignment is displayed in the "Current Keys" field.
4. Position the cursor in the "Press New Shortcut Key" field and press the new shortcut key(s) on your keyboard.
   The keys are displayed in the field.
5. Choose the **Assign** button.
   The assignment is entered into the "Current Keys" field.

When you close the "Customize" dialog, the new key assignment will not replace the existing assignment, but will be available **in addition to** it. You can remove the existing assignment from the "Current Keys" field.

Removing a Key Assignment

To remove an assignment

1. Select the assignment to be removed in the "Current Keys" field.
2. Choose the **Remove** button and choose **Yes** to confirm the reset confirmation query.

Resetting Your Personal Key Assignments

To undo all personal key assignments

- Choose the **Reset All** button.

User Commands

The "User Commands" sheet in the "Customize..." dialog enables you to specify user-defined commands.

The following topics are covered below:

- Invoking the User Commands Dialog
- Defining a User Command
- Adding a User Command to an Existing Toolbar

Invoking the User Commands Dialog

To invoke the dialog

1. From the **Tools** menu, select "Customize...".
2. Click on the "User Commands" tab.
   The "User Commands" dialog box appears.

Defining a User Command

You can define commands which contain a series of Natural commands. These user-defined commands can be added to any toolbar.

To define a user command

1. In the "User Commands" dialog box, select the user command to be defined, for example:
   `User Command 1`.
2. In the "Natural Command(s)" field, enter the Natural command(s) that are to be invoked with `User Command 1`.
   Separate multiple commands with a semicolon.
3. Click on **Assign** to assign the Natural command(s) to the user command.
   The command assigned is shown next to the user command entry in the "User Command" dialog box.
4. Click on the **Close** button.
   The user command assignment is added to the command list from where it can be added to an existing toolbar.

Adding a User Command to an Existing Toolbar

To add a user-defined command to a toolbar

- Click to the "Commands" tab of the "Customize..." dialog.
- From the "Categories" box, select "User Commands".
  The list of user commands is displayed.
- From the "Commands" list, select the user command and drag it to the toolbar at the top of the screen.
- Choose **Close**.

You can move the corresponding icon to another position in the toolbar, as long as the "Customize..." dialog is open.

Tutorial - Getting Started with Natural

This tutorial is designed to provide a basic understanding of specific features of the Natural programming environment and illustrates how an application can be structured as a group of modules. It is **not** intended to provide an example of how an application should be built.

These sessions also represent a general introduction to how the editors can be used. Therefore explanations are kept to a minimum. This tutorial is **not** intended to be a comprehensive description of the full range of possibilities provided by the Natural editors. For a full description of all editor functions and features, please refer to the corresponding sections in this documentation:

Program Editor | Data Area Editor | Map Editor | DDM Editor | Dialog Editor

Prerequisite:
To perform all steps of this tutorial, the database SAG-DEMO-DB must be installed and active. To start the database, double click the SAG-DEMO-DB icon in the Natural program group.

- Session 1 - Creating and Modifying a Program
- Session 2 - Creating and Editing a Map
- Session 3 - Checking and Running a Program
- Session 4 - Creating a Local Data Area
- Session 5 - Creating a Global Data Area
- Session 6 - Creating an External Subroutine
- Session 7 - Invoking a Subprogram

Session 1 - Creating and Modifying a Program

In this session, you will create and save a Natural program, using the program editor to enter source statements in a program editor window.

Step 1

Natural user-written applications are stored in libraries. It may be necessary to move from one library to another in order to perform a maintenance function or work on a different application. The application created in these sessions will be stored in the SYSEXPG library.

Select the SYSEXPG library node in the library workspace.

To open the library SYSEXPG

1. From the tree view, choose "System Libraries".
2. Scroll to SYSEXPG and select it.

Step 2

Natural offers two modes of programming: structured mode and reporting mode.
Software AG recommends that you use structured mode exclusively, because it results in more clearly structured applications. Therefore all explanations and examples in this chapter refer to structured mode. Any properties of reporting mode will not be taken into consideration. You must be operating in structured mode to work through the sessions in this chapter.

If the current mode is reporting mode, change it to structured mode:

To do so

1. From the **Tools** menu, choose **Session Parameters** > **Compiler options**.
2. Select "Structured Mode".
3. Choose **OK**.

Step 3

The SYSEXPG library should include the program used in this session, PGM01. In this step, you will either edit or create the program.

Edit PGM01

If PGM01 is available, edit the program.

To open PGM01 for editing

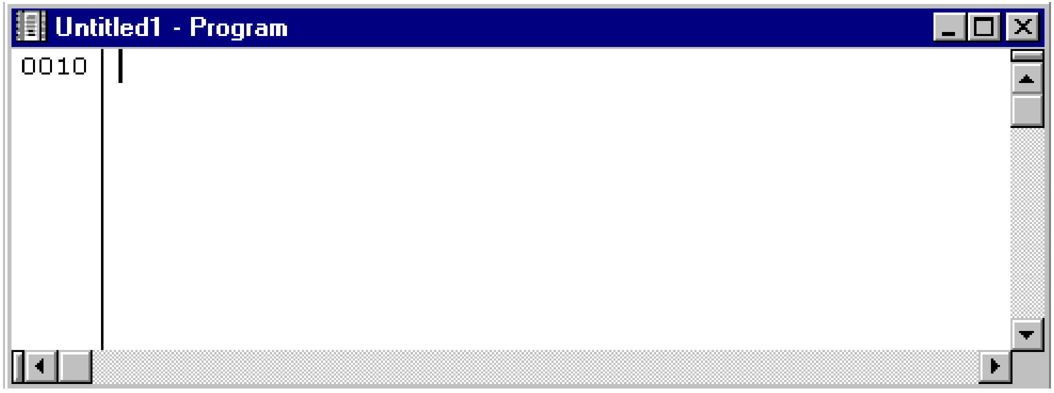
- Expand the library node, expand the "Programs" node, select the Program PGM01 and press ENTER.

Create PGM01

If PGM01 is not available, you can create it.

To open a new program editor window

- Open the context menu of the "Programs" node and select the **New** item.
  The program editor window is displayed:



To modify "Program Editor Options"

If line numbers are not usable, you can modify them.

1. From the main menu bar select **Tools** > **Options** > **Program Editor**.
2. Set "Line Numbers" check box.

Step 4

To save the program under the name "PGM01"

1. From the **Object** menu, choose **Save**.
   If the program already exists in the library, then it is saved. Go to Step 5.
   If the program does not yet exist in the library, the "Save as" dialog box appears.
2. In the "Name" text box, enter "PGM01".
3. Choose **OK**.

The program is now saved under the name "PGM01" in the library SYSEXPG.

Step 5

To close PGM01 before ending the session

- From the **Object** menu, select **Close** or press **CTRL-F4**.

End of Session 1.

Session 2 - Creating and Editing a Map

The Natural map editor is used for creating the maps referenced in a Natural program. Once a map has been created, it can be stored in the Natural system file, where it can be invoked by a Natural program using a WRITE or INPUT statement.

A map consists of fields. A field can be a text field (a constant) or a data field (a variable), or any of the graphical user interface elements provided in the map editor’s "Insert" menu. The fields that comprise a map can be defined direct in the map editor window, or imported from another source object, such as a DDM, a program, or a data area. Natural system variables can be imported as well.

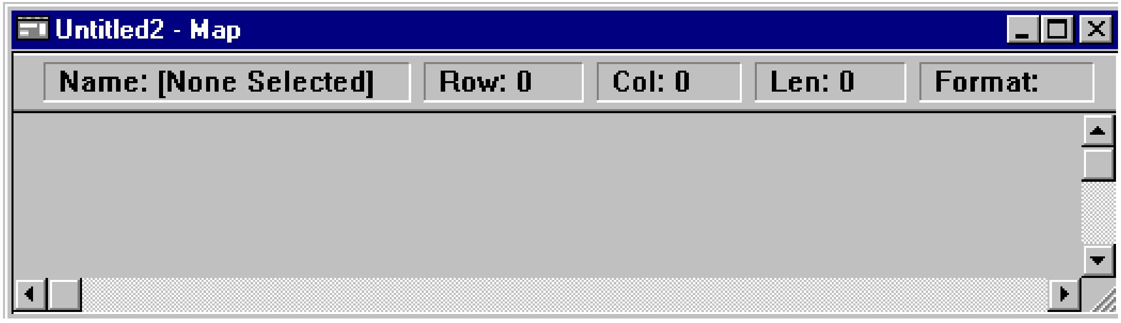
In this session, you will create a map that contains text fields, data fields, and system variables.

Step 1

In the previous session, the screen prompting for an employee name was produced through the INPUT USING MAP statement using MAP01. In this session, you will create the map. Note that in the INPUT USING MAP statement, the map must be specified in quotation marks to distinguish the map from a user-defined variable.

To open a new map editor window

- Open the context menu of the SYSEXEVT node and select the **New** > **Map** item.
  The map editor window appears.



Step 2

A text field is a constant that you create using the text field entry in the **Insert** menu, or that you import from another Natural object. Its format is always A (for alphanumeric).

You can create a title for the map by drawing a text field and defining the text it will contain.

To do so

1. From the **Insert** menu, choose **Text Constant**.
   Or click the **Text Constant** toolbar button.
2. Place the text field at the top of the editor, where you want the field to begin.
3. Draw a field by holding down the left mouse button and dragging the mouse to the right about half the width of the editor.
   The text field you have just drawn is still selected. When a field is selected, its field handles appear.

The field must be selected before you can perform many of the map editor functions, such as defining a field and selecting a color for the field.

To define the text field

1. Point to the field and double-click.
   Or, from the **Field** menu, choose **Definition**.
   In the text field, you can now enter the text.
2. Type "SOFTWARE AG EMPLOYEE INFORMATION".
   Select the field again by clicking the mouse with the pointer outside the field and then with the pointer on the field.

To select a color for the text field

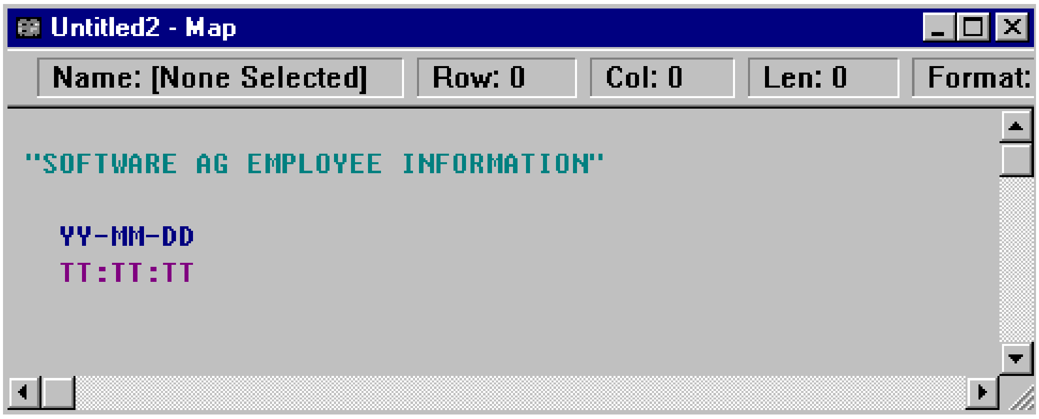
1. From the **Field** menu, choose **Color**.
   Or click the **Field Color** toolbar button.
2. Select any color you want for this field. (Click the name of the color or its **Option** button.)
3. Choose **OK**.
   "SOFTWARE AG EMPLOYEE INFORMATION" appears on the map in the color you selected.
4. To deselect the field, move the pointer away from the field and click.
   The field handles disappear.

Step 3

Natural system variables can be imported into a map. The system variables *DATX and *TIMX display the current date and time, when the program that invokes the map is executed.

To import the *DATX system variable

1. From the **Insert** menu, choose **Import**.
2. Choose **System variable**. The "Import System Variable" dialog box appears.
3. Scroll to *DATX and select it.
4. Choose **Import**. The system variable will appear in the top left corner of the map.
5. Choose **Quit** to close the dialog box.
6. Move the *DATX field cursor below SOFTWARE AG EMPLOYEE INFORMATION.
7. Select a color for the *DATX field.
   Import the *TIMX system variable. Use the same procedure you used to import the *DATX system variable. Select a color for the *TIMX field, then move *TIMX to the line below *DATX.
   The map should now look as follows.



Name: [None Selected] Row: 0 Col: 0 Len: 0 Format:

"SOFTWARE AG EMPLOYEE INFORMATION"

YY-MM-DD
TT:TT:TT

Step 4

New fields can be created by copying and redefining existing fields.

To copy a field to the clipboard

1. Select the text field "SOFTWARE AG EMPLOYEE INFORMATION".
2. From the **Edit** menu, choose **Copy**.

To paste the copied field into the map

1. From the **Edit** menu, choose **Paste**.
2. Drag the copied field from the top left corner to below the *TIMX field.
   Notice that this field is the same color as the field you copied. If you want to change its color, from the **Field** menu, select **Color**.

To define the new field in the "Text Field Definition" dialog box

1. Point to the field and double-click.
   Or, from the **Field** menu, choose **Definition**.
   In the text field, you can now enter the text.
2. Type "PLEASE ENTER STARTING NAME:".

Step 5

A data field is a field that you create using the Data field entry in the "Insert" menu, a field that you import from another Natural object, or a Natural system variable.

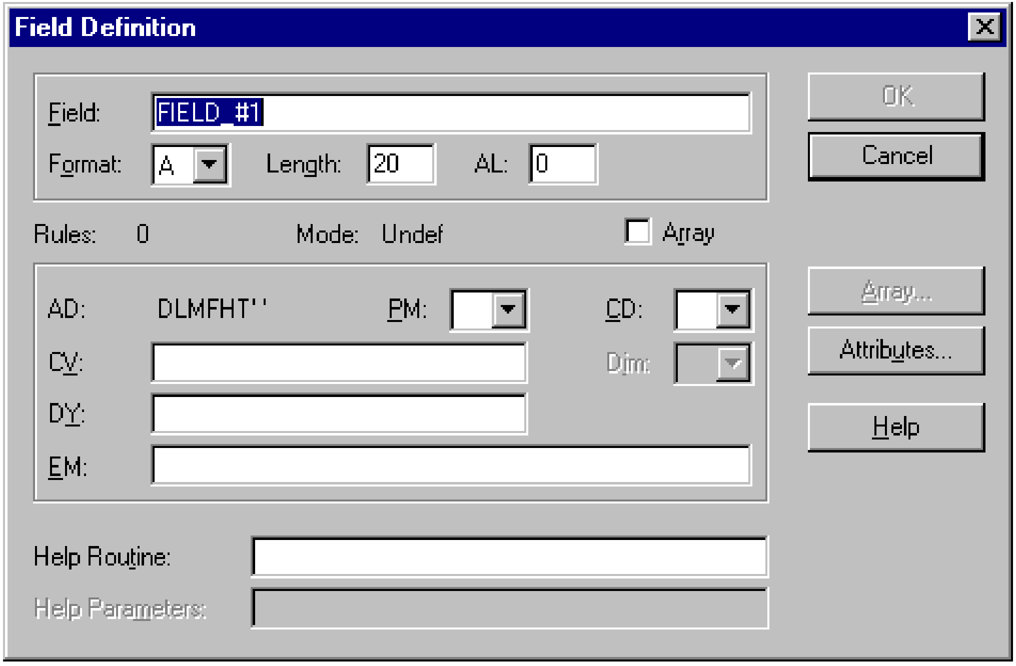
In this step, you will draw a data field and define its attributes.

To draw the data field

1. From the **Insert** menu, choose **Data Field**.
   Or click on the **data field drawing tool** toolbar button.
2. Place the data field to the right of PLEASE ENTER STARTING NAME:
3. Draw a field that is 20 characters long. (Using the mouse, drag the data field across the map until Len=20).

To define the data field

1. Point to the field and double-click.
   Or from the **Field** menu, choose **Definition**.
   The "Field Definition" dialog box appears:

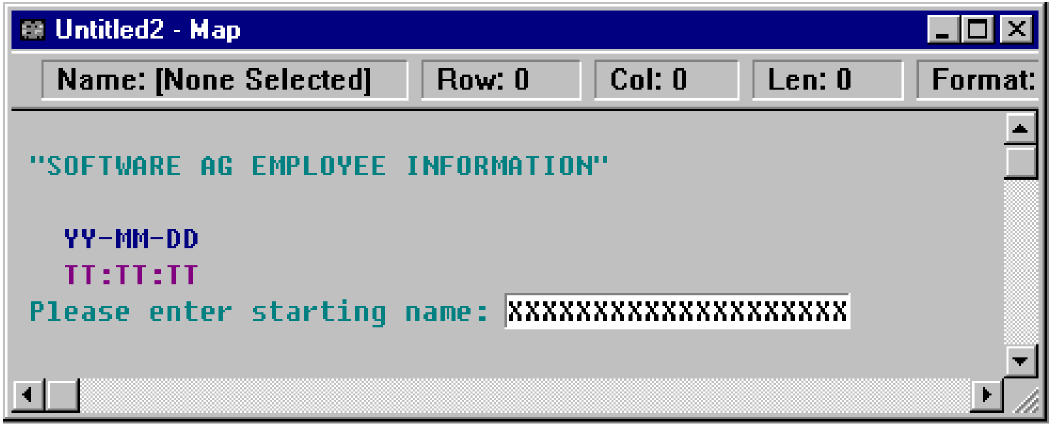


Field: FIELD_#1
Format: A Length: 20 AL: 0
Rules: 0 Mode: Undef □ Array
AD: DLMFHT'' PM: CD:
CV: Dim:
DY:
EM:
Help Routine:
Help Parameters:

2. In the "Field" text box, delete the name and type in "#NAME-START". Press the TAB key.
   Format "A" (alphanumeric) is the correct format for this field.
   The alphanumeric length of the field should be "20". If not, use TAB to move the cursor to the "Length" field and enter "20".

To specify attributes for the data field

1. Choose **Attributes**.
   The "Attribute Definitions" dialog box appears.
2. Select the "I/O Characteristics" list box and select "Output, Modifiable" to define the field as an output field that can be modified.
3. Enter underscore () as filler character.
   This is the character that is used to fill any empty positions in input fields in the map, allowing the user to see the exact position and length of a field when entering input.
4. Choose **OK**.
   The "Field Definition" dialog box is displayed again.
5. Choose **OK** to save the data field definition that you entered.
   The map could now look as follows:



```
"SOFTWARE AG EMPLOYEE INFORMATION"

  YY-MM-DD
  TT:TT:TT
Please enter starting name: XXXXXXXXXXXXXXXXXXXX
```

Step 6

In this step, you will edit the map to add an ending name for a range of employees.

In the same way as you have created text fields and data fields so far, draw and define another text field and another data field.

To draw and define the "PLEASE ENTER ENDING NAME:" text field

1. Choose **Insert** > **Text Constant** to create the text field and draw a field 25 characters long, one line below "PLEASE ENTER STARTING NAME:"
2. In the text field, enter PLEASE ENTER ENDING NAME:
3. Select a color for the text field.

To draw and define the data field "#NAME-END"

1. Choose **Insert** > **Data Field** to draw a field 20 characters long, one space to the right of the text constant.
2. In the "Data Field Definition" dialog box, enter "#NAME-END" as the field name (the format is "A" and the length is "20").
3. Choose **Attributes** and select "Output, Modifiable" as the I/O Characteristic.
4. Choose **OK** twice.
5. Select a color for the "#NAME-END" data field.

The output of this data field is a user-defined variable found in the DEFINE DATA statement of PGM01 that will correspond to the new field definition entered on the map.

Step 7

To center the field "SOFTWARE AG EMPLOYEE INFORMATION" at the top of the map

1. Select the field.
2. From the **Field** menu, choose **Alignment**.
3. From the cascading menu, choose **Map center**.
   The text moves to the center of the map.

To move fields to different locations in the map:

1. Move the "*DATX" field to the top line of the map (Row=1).
2. Move the "*TIMX" field to line three (Row=3), directly below the "*DATX" field.

Step 8

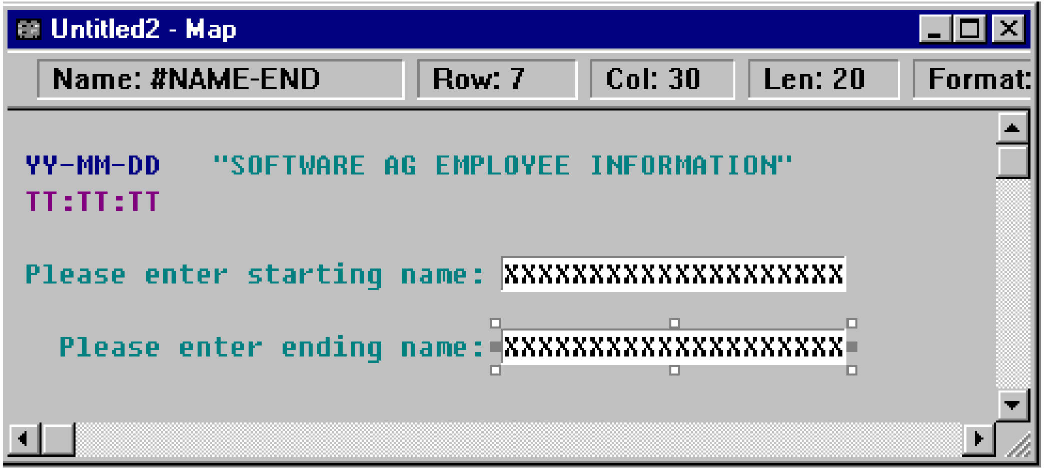
In this step, you will move ranges of fields to new locations.

To position the first range of fields

1. Select a range of fields that contains the text field "PLEASE ENTER STARTING NAME:" and the "#NAME-START" data field.
   Select the fields by holding down the left mouse button and dragging the mouse to surround the fields. Release the mouse button to select the fields.
2. Move the range to line five of the map (Row=5).
   Move the range by placing the selector tool within the field handles and dragging the range to the new location.

To position the second range of fields

1. Using the same method as above, select the text field "PLEASE ENTER ENDING NAME:" and the data field "#NAME-END".
2. Move the range to Line Seven of the map (Row=7).
   The map now looks as follows.



Untitled2 - Map

Name: #NAME-END | Row: 7 | Col: 30 | Len: 20 | Format:

```
YY-MM-DD   "SOFTWARE AG EMPLOYEE INFORMATION"
TT:TT:TT

Please enter starting name: XXXXXXXXXXXXXXXXXXXX

  Please enter ending name: XXXXXXXXXXXXXXXXXXXX
```

Step 9

The first time you save a map, you must give it a name. After the map is named, you can make changes to it and save it or stow it without entering the name. If you want to save a modified map with a different name, choose "Save as" and enter a different name.

To save the map and give it a name

1. From the **Object** menu, choose **Save As**.
   The **Save As** dialog box appears. The current library is SYSEXPG, the library where the map is saved.
2. In the "Name" text box, type "MAP01".
3. Choose **Save**.
   The map is saved as MAP01 in the SYSEXPG library.

Step 10

Now, you will create a processing rule for a map field.

To define a processing rule for the #NAME-START data field

1. Click the #NAME-START field once to select it.
2. From the **Field** menu, choose **Rules**.
   The "Field Rules" dialog box appears.
3. Choose **Create**.
   A program editor window opens. Enter the following processing rule:

```
IF & = ' ' REINPUT 'PLEASE TYPE IN A NAME'
  MARK *&
END-IF
*
```

Note:
The ampersand (&) in the processing rule will be dynamically replaced by the name of the field.

To save the processing rule and give it a rank

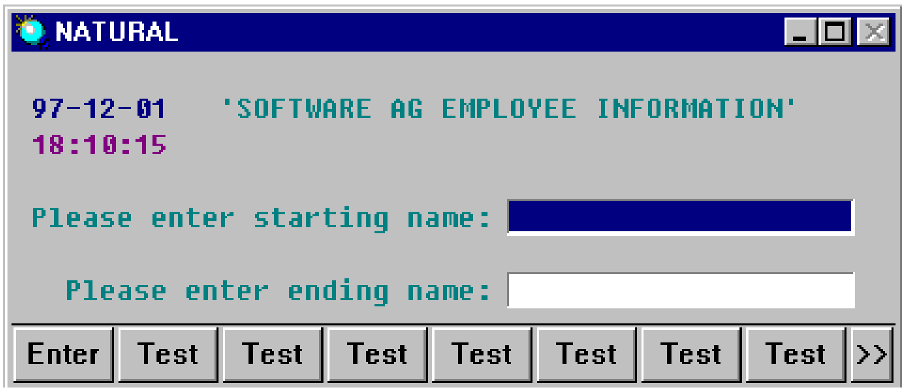
1. From the **Object** menu, choose **Save as**. The "Rule Selection" dialog box appears.
2. From the list box, select "1", and then choose **OK**.
3. Close the "Map Rule" (program editor) window by choosing **Close** from the **Object** menu.
   The window closes and MAP01 reappears.

Step 11

In this step, you will test MAP01 to check whether it works as intended.

To test the map

1. From the **Object** menu, choose **Test**.
   The map, including the processing rule, is executed. This is the same screen that appears when the map is invoked from PGM01:



```
97-12-01    'SOFTWARE AG EMPLOYEE INFORMATION'
18:10:15

Please enter starting name:

  Please enter ending name:
```

2. Type in a name and press ENTER.
   You are returned to the map editor.
   When you do not enter a name and press ENTER, the message "Please enter starting name" is displayed in the message line.

Step 12

When the map has been successfully tested, it has to be stowed; that is, stored in both source and object form.

To stow the map

- From the **Object** menu, choose **Stow**.

Step 13

The next step is to create a helproutine and attach it to a field in a map.

To modify the field definition for the field "#NAME-START"

1. Select the field "#NAME-START".
2. Either select **Definition** from the **Field** menu or point to the "#NAME-START" field and double-click.
   The "Field Definition" dialog box appears.
3. Use TAB to move to the "Help Routine:" text box, and enter "’HELP001’" (do not forget the quotation marks).
   "HELP001" (which is yet to be created) is the name of the helproutine that is invoked when a user presses the HELP key while the cursor is in the "#NAME-START" field.
4. Choose **OK**.
   The map editor window appears.
5. Stow the map (that is, store it in source and object form) by choosing **Stow** from the **Object** menu.
6. From the **Object** menu, choose **Close**.

Step 14

Now the helproutine itself has to be created.

To create the helproutine

- Open the context menu of the "SYSEXEVT" node and select the **New** > **Helproutine** item.

End of Session 2.

Session 3 - Checking and Running a Program

In the previous session, you added a variable called #NAME-END to MAP01. This variable allows the program to provide an ending point for the READ statement. Otherwise, all employees from JONES to the end of the alphabet would be included in your report.

Now that the map allows both the beginning and ending name to be provided on the input screen, an IF statement must be added to the PGM01 program.

Step 1

Make sure that SYSEXPG is the current library.

In the "Programs" folder, scroll to "PGM01" and select it.

The program editor is invoked and the current version of the program PGM01 appears.

For easier editing, you can maximize the program editor window by clicking the "Maximize" button.

Step 2

The program includes the following statement:

```
MOVE #NAME-START TO #NAME-END
```

Replace this statement with the following IF statement:

```
IF #NAME-END = ' '
  MOVE #NAME-START TO #NAME-END
END-IF
```

Step 3

You can add user comments to a program to identify the program modifications that you have made. A user comment helps anyone editing or maintaining a source program and is ignored during processing.

A user comment is entered by inserting a statement line or lines. If the entire line is to be reserved for a user comment, enter an asterisk and a blank (*) or two asterisks (**) in columns 1 and 2 of the line and type in the comment. If you want to place a comment in the same line of source code, separate the code from the comment with " /*" (a blank, a slash and an asterisk).

Add a comment to Line 3 to indicate that the program has been modified, for example:

```
 * A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
```

Step 4

When you have completed the above modifications to PGM01, the program should look as follows:

```
* PGM-ID:  PGM01
* FUNCTION:   DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* -----------------------------------------------------
DEFINE DATA
  LOCAL
  01  #NAME-START           (A20)
  01  #NAME-END             (A20)
  01  #MARK                 (A1)
  01  EMPLOYEES-VIEW        VIEW OF EMPLOYEES
      02  PERSONNEL-ID      (A8)
      02  NAME              (A20)
      02  DEPT              (A6)
      02  LEAVE-DUE         (N2)
END-DEFINE
*
REPEAT
*
  INPUT USING MAP ’MAP01’
*
  IF #NAME-START = ’.’
    ESCAPE BOTTOM
  END-IF
*
  IF #NAME-END = ’ ’
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW
          BY NAME
          STARTING FROM #NAME-START
          THRU #NAME-END
*
    IF LEAVE-DUE >= 30
       PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
       RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X ’>=30’ #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT ’PLEASE TRY ANOTHER NAME’
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE ’*’ TO #MARK
END-SUBROUTINE
*
END
```

Save the modified version of PGM01 by choosing "Save" from the "Object" menu.

Step 5

Checking a program allows you to find and correct syntax errors that would otherwise prevent the program from being compiled. In this step, you will create an error in the source code of PGM01. Then you will check the program to identify the error, correct the error, and run the program.

To create an error in the PGM01 source code

1. Edit PGM01.
2. Use the arrow keys or the **Go to** function of the **Edit** menu to move the cursor to the following line:
   `DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK`
3. Move the cursor to the second quotation mark and press **DEL** to remove the quotation mark.
   Natural uses beginning and ending quotation marks to designate text strings. A text string must be closed on the same line in which it was opened. When the Natural compiler finds an odd number of quotation marks on the same line, then it reports a syntax error.
4. From the **Object** menu, choose **Check**.
   When the error is detected, syntax checking is suspended. The line that contains the error is displayed, and the following error message appears:
   `NAT0305 TEXT STRING MUST BEGIN AND END ON SAME LINE`

To correct the error and check the program again

1. Add an quotation mark directly, and press the **CONTINUE** button.
   Or press **ENTER** to return to the program editor and make the correction.
2. From the **Object** menu, choose **Check**.
   When the syntax error has been corrected, and if no other syntax errors are detected, you are informed that the check was successful.
3. Choose **OK**.

Step 6

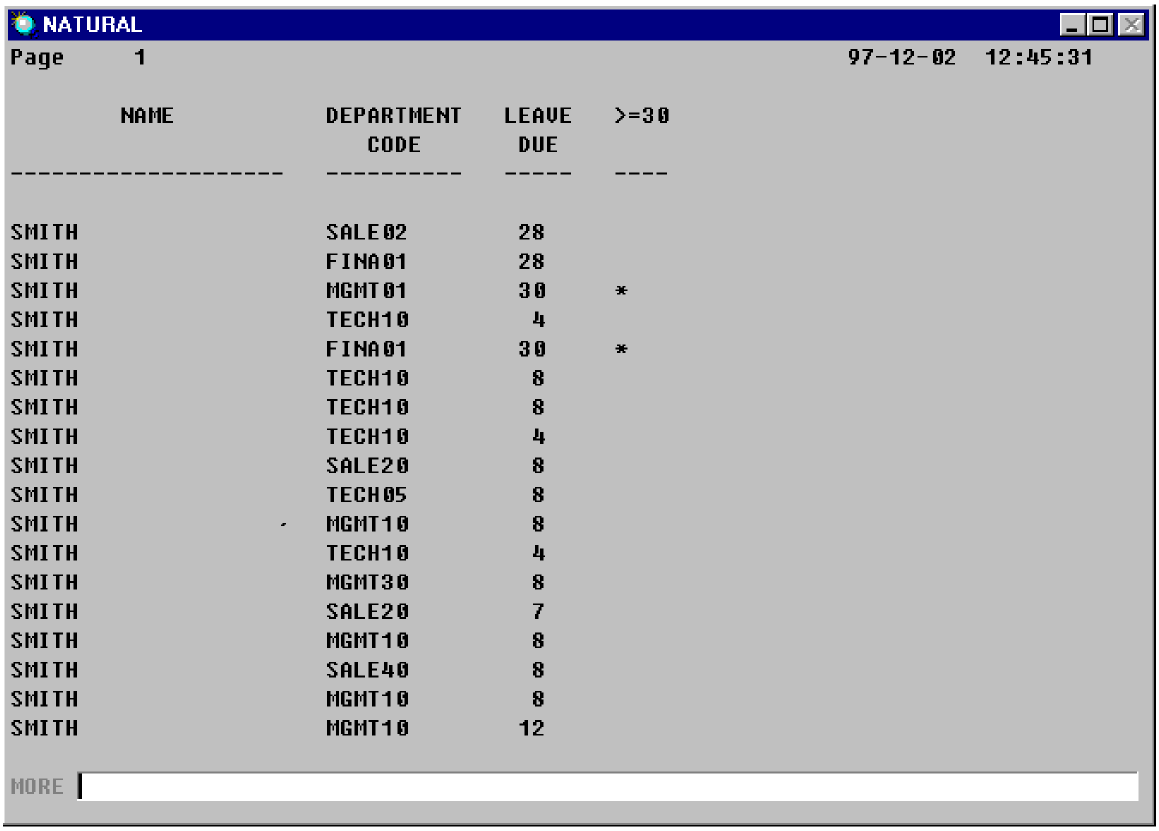
In this step, you will run the program PGM01 and view the output. When you run this program, you are prompted to enter a name. The EMPLOYEES file is searched to locate all employees with that name; then a report that includes the Name, Department and Leave Due to each employee with that name is displayed. The names of employees who have 30 or more days leave due are marked with an asterisk.

The prompting screen is invoked at the INPUT USING MAP statement. The final report is formatted according to information in the DISPLAY statement.

The processing required to show which employees have more than 30 days leave is handled in the portion of the program starting with IF LEAVE-DUE. Those with 30 or more days of leave due have an asterisk in the final report as a result of processing in the PERFORM statement and the DEFINE SUBROUTINE statement.

To see if everything - including the map and the helproutine - works as intended

1. From the **Object** menu, choose **Run** to compile and execute the program PGM01.
   The map MAP01 is displayed.
2. Press **ENTER** without typing in anything.
   The following message is displayed:
   `PLEASE TYPE IN A NAME`
3. In the first input field in the map, enter a question mark (?).
   The helproutine HELP001 appears:
   `TYPE THE NAME OF AN EMPLOYEE.`
4. In the first input field of the map, type the name MCKENNA, and press **ENTER**.
   As there is no record with the name MCKENNA in the database, the following message is displayed:
   `PLEASE TRY ANOTHER NAME`
5. In the first input field of the map, type the name SMITH, and press **ENTER**.
   The database does include the name SMITH; the following list is displayed:

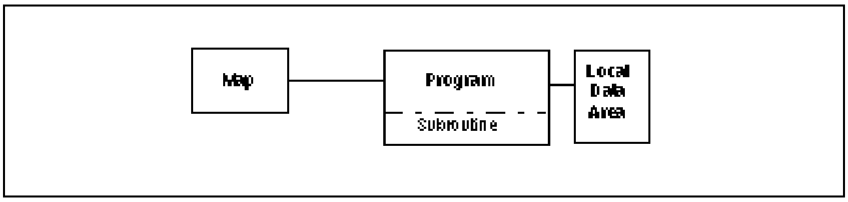


```
Page      1                                                     97-12-02   12:45:31

        NAME              DEPARTMENT   LEAVE   >=30
                             CODE       DUE
--------------------    ----------   -----   ----

SMITH                   SALE02        28
SMITH                   FINA01        28
SMITH                   MGMT01        30      *
SMITH                   TECH10         4
SMITH                   FINA01        30      *
SMITH                   TECH10         8
SMITH                   TECH10         8
SMITH                   TECH10         4
SMITH                   SALE20         8
SMITH                   TECH05         8
SMITH                   MGMT10         8
SMITH                   TECH10         4
SMITH                   MGMT30         8
SMITH                   SALE20         7
SMITH                   MGMT10         8
SMITH                   SALE40         8
SMITH                   MGMT10         8
SMITH                   MGMT10        12

MORE
```

6. Press **ENTER**.
7. When the program prompts you again for a name, enter a period (.). Press **ENTER** again to return to the program editor window.
8. Close PGM01.

End of Session 3.

Session 4 - Creating a Local Data Area



In Session 1, the fields used by the program were defined within the DEFINE DATA statement in the program itself. It is also possible, however, to place the field definitions in a local data area outside the program, with the program’s DEFINE DATA statement referencing that local data area by name. For a clear application structure, it is usually better to define fields in data areas outside the programs.

In this session, the information in the DEFINE DATA statement will be relocated to a local data area outside the program. In subsequent sessions, some of this information can be used as the basis of a global data area shared by a program and an external subroutine. As you will see later in this tutorial, an important advantage of data areas is to allow a program and its external subroutine to share the same data in a single data area.

Step 1

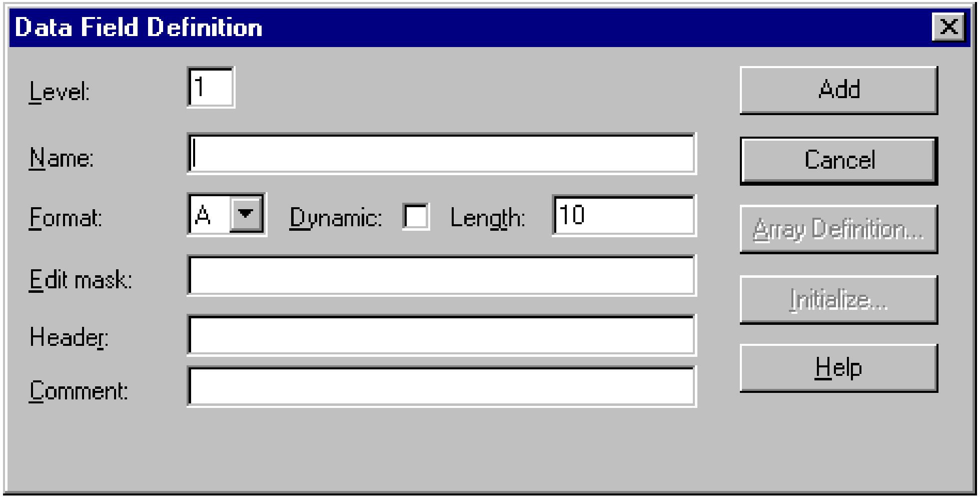
In this step, you will create a data area with three data fields. Each data field must be defined separately.

To open a local data area editor window

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Local Data Area**.

To insert the first data field

1. From the **Insert** menu, choose **Data Field**.
   The "Data Field Definition" dialog box is displayed.



Level: 1
Name:
Format: A Dynamic: □ Length: 10
Edit mask:
Header:
Comment:

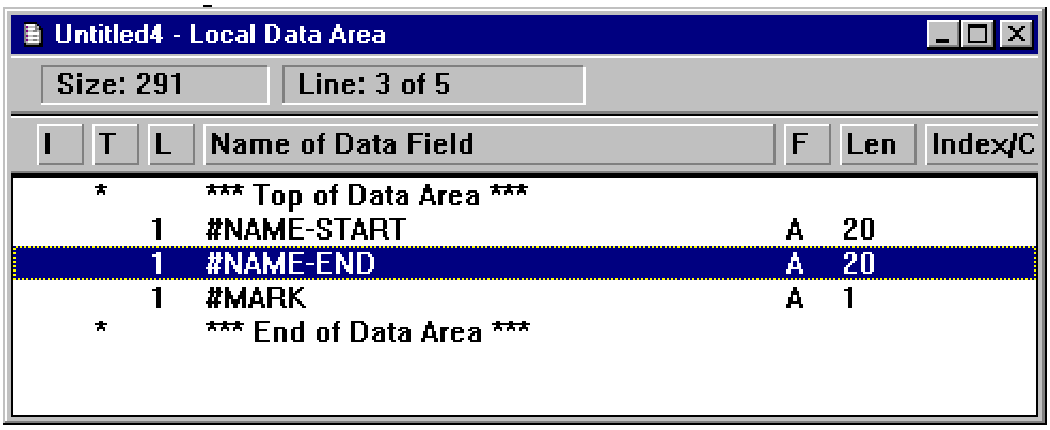
   In the "Level" text box the default "1" is displayed.
2. In the "Name" text box, enter "#NAME-START".
3. Format "A" is the correct format for the "#NAME-START" data field. (Alphanumeric is the default format).
4. In the "Length" text box, enter "20".
5. Choose **Add.**
   The "Define a Data Field" dialog box appears again to allow you to define another data field.

Define a second and third data field with the following attributes:

Field Name	**Data Field 2**	**Data Field 3**
Level:	1	1
Field:	#NAME-END	#MARK
Length:	20	1
Format:	A	A

When the "Data Field Definition" dialog box is displayed again, choose **Quit** to end the field definition process.

The local data area now looks as follows:



Size: 291 Line: 3 of 5

I	T	L	Name of Data Field	F	Len	Index/C
	*		*** Top of Data Area ***			
		1	#NAME-START	A	20	
		1	#NAME-END	A	20	
		1	#MARK	A	1	
	*		*** End of Data Area ***			

Step 2

To confirm that no syntax errors have been made

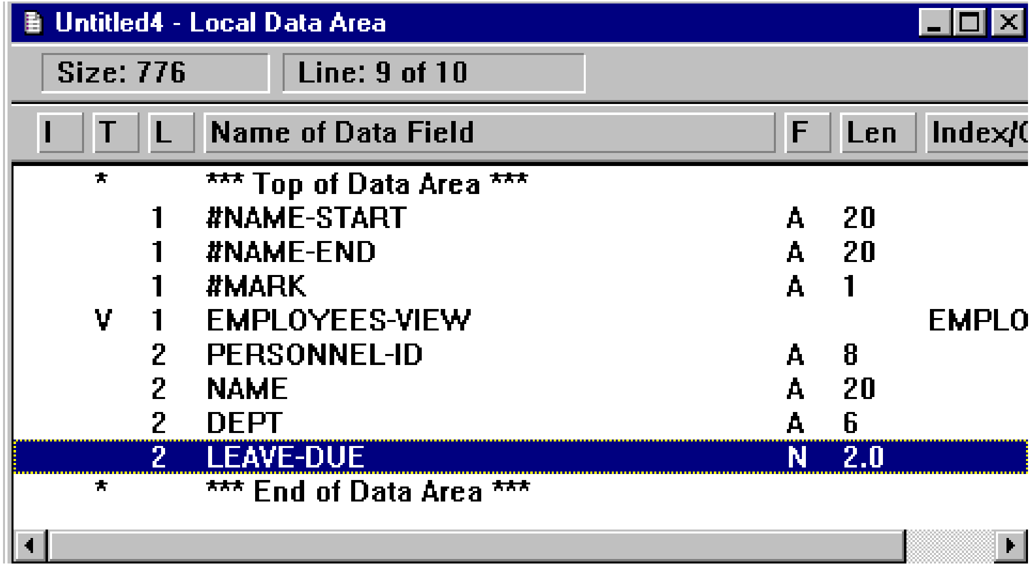
- From the **Object** menu, choose **Check**.

Step 3

Variables defined in a Natural DDM can be imported directly into the local data area.

To import fields from the "EMPLOYEES" DDM

1. Select the "#MARK" field.
2. From the **Insert** menu, choose **Import**.
   The "Import View" dialog box appears with the name of the current library (SYSEXPG) in the "Library" list box.
3. Open the "Library" list box and select the SYSEXDDM library.
   A list of all DDMs in the SYSEXDDM library appears in the DDM list box.
4. Select the "EMPLOYEES" DDM.
   A list of all the data fields in the "EMPLOYEES" DDM appears in the "Data Fields" list box.
5. Scroll through the list and select the following fields: "PERSONNEL-ID", "NAME", "DEPT", and "LEAVE-DUE".
   Note:
   To select individual fields, hold down **CTRL** while you click the left mouse button.
6. Choose **OK**.
   The "View Definition" dialog box appears.
7. Enter "EMPLOYEES-VIEW" as the name of the view.
8. Choose **OK**.
   The imported fields appear in the local data area, after the "#MARK" field. The name of the view that contains these fields (EMPLOYEES-VIEW) also appears in the data area and is identified with a V in the T (Type) column.



Size: 776 Line: 9 of 10

I	T	L	Name of Data Field	F	Len	Index/(
	*		*** Top of Data Area ***			
		1	#NAME-START	A	20	
		1	#NAME-END	A	20	
		1	#MARK	A	1	
	V	1	EMPLOYEES-VIEW			EMPLO
		2	PERSONNEL-ID	A	8	
		2	NAME	A	20	
		2	DEPT	A	6	
		2	LEAVE-DUE	N	2.0	
	*		*** End of Data Area ***			

Step 4

To check the new local data area

1. From the **Object** menu, choose **Check**.
2. If syntax errors are found, correct them; then check the local data area again.

Step 5

To stow the new local data area

1. From the **Object** menu, choose **Stow**.
   The "Stow As" dialog box appears.
2. In the "Name" text box, enter "LDA01".
   As the library SYSEXPG is highlighted in the "Library" list box, the LDA01 local data area will be stored in this library.
3. Choose **OK**.

Step 6

To close the LDA01 local data area before continuing this session

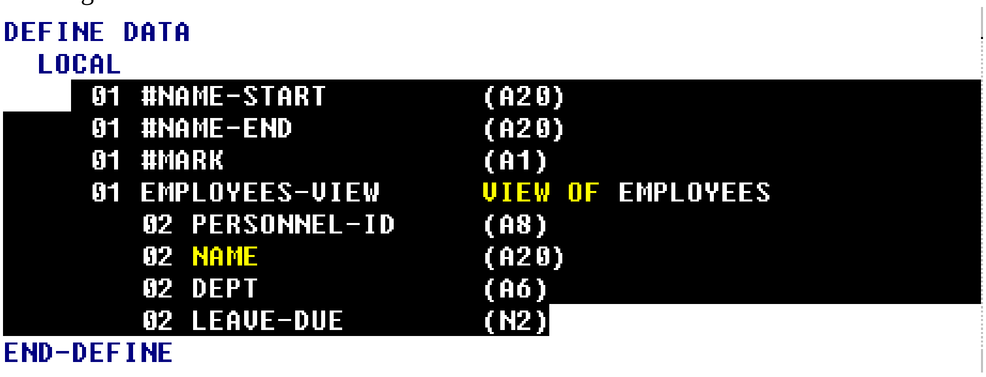
- From the **Object** menu, choose **Close**.

Step 7

In this step, the PGM01 program is modified to reference the LDA01 local data area. After removing the lines within the DEFINE DATA statement that define variables, you will add a statement to reference the local data area.

To edit PGM01

1. Open the SYSEXPG library and then, from the "Objects" window, open the program PGM01.
2. Maximize the program editor window for easier editing.
3. Remove the lines that define variables:
   Place the cursor at the beginning of the line containing "#NAME-START" and use the mouse to select the following text:



```
DEFINE DATA
  LOCAL
    01 #NAME-START         (A20)
    01 #NAME-END           (A20)
    01 #MARK               (A1)
    01 EMPLOYEES-VIEW      VIEW OF EMPLOYEES
       02 PERSONNEL-ID     (A8)
       02 NAME             (A20)
       02 DEPT             (A6)
       02 LEAVE-DUE        (N2)
END-DEFINE
```

4. From the **Edit** menu, choose **Delete**.
5. Add a reference to LDA01 by entering the following statement in the blank line after LOCAL:
   USING LDA01

The program should now look as follows:

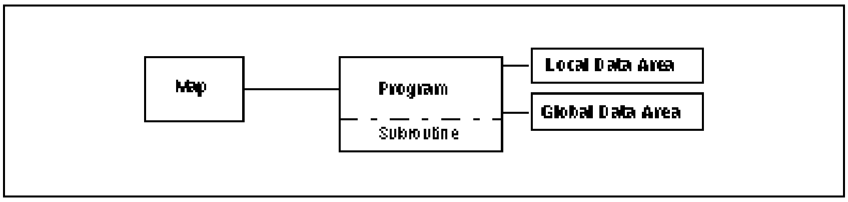
```
* PGM-ID:    PGM01
* FUNCTION:    DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* PROGRAM NOW USES A LOCAL DATA AREA
* -----------------------------------------------------
DEFINE DATA
    LOCAL
      USING LDA01
END-DEFINE
*
REPEAT
*
    INPUT USING MAP 'MAP01'
*
  IF #NAME = '.'
    ESCAPE BOTTOM
  END-IF
*
  IF #END = ' '
    MOVE #NAME TO #END
  END-IF
*
  RD1.  READ EMPLOYEES-VIEW
          BY NAME
          STARTING FROM #NAME
          THRU #END
*
    IF LEAVE-DUE >= 30
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
 DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

Step 8

1. Check the PGM01 program and correct any errors.
2. Run PGM01 to confirm that the results are the same as when the DEFINE DATA statement did not reference a local data area.
3. Stow PGM01 so that it is available for Session 5.
4. Close PGM01.

End of Session 4.

Session 5 - Creating a Global Data Area



In Natural, data can be defined in a single location outside any particular program or routine. Data defined in such a global data area can then be shared by multiple programs/routines.

In this session, you will create a global data area. In addition, you will modify the local data area created in the previous session. You will also modify the program so that it references not only the local data area, but also the new global data area.

Step 1

The local data area that you created in Session 4 (LDA01) is stored in the SYSEXPG library. Before you start this session, make sure that the SYSEXPG library is the current library.

You can create a new data area from an existing data area by editing the data area and saving it with a different name and type. The original data area remains unchanged, and the new data area can be edited.

In this step, you will use the local data area LDA01 to create a global data area.

Open LDA01.

To save LDA01 with the name "GDA01" and change the type to "GDA"

1. From the **Object** menu, choose **Save As**.
   The "Save As" dialog box appears.
2. In the "Name" text box, enter GDA01.
   Do not change the name of the current library (SYSEXPG). The new global data area is stored in the SYSEXPG library.
3. Open the "Type" list box and select "Global".
4. Choose **OK**.
   The data area is saved as a global data area named "GDA01". GDA01 appears in the data area editor window.

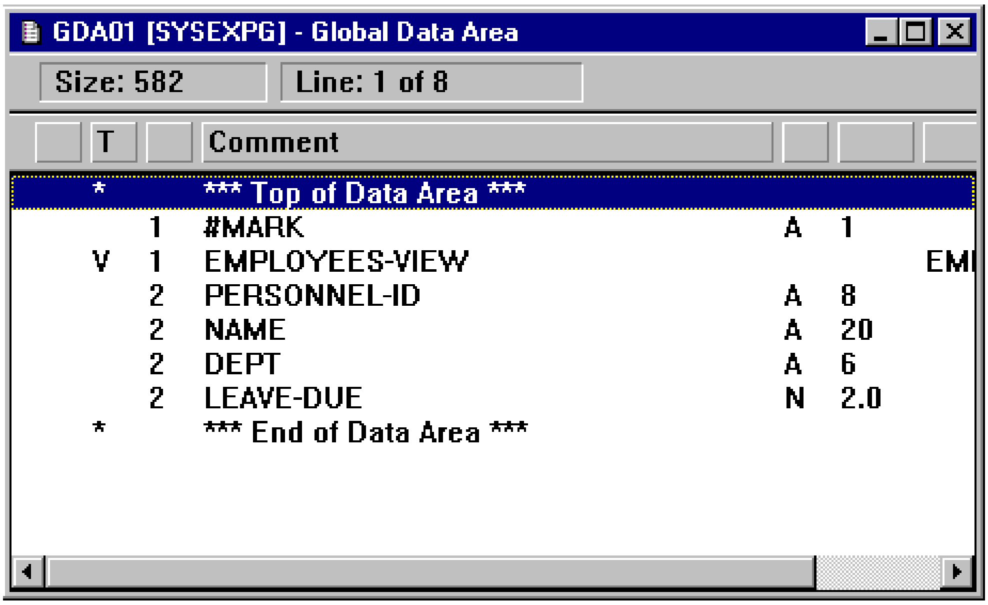
Step 2

To remove the data fields "#NAME-START" and "#NAME-END"

1. Select the fields "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.

Note:
To select multiple fields, hold down the left mouse button and drag the mouse across the fields to be selected.

The global data area should now look as follows:



GDA01 [SYSEXPG] - Global Data Area

Size: 582 Line: 1 of 8

```
     T        Comment
     *        *** Top of Data Area ***
         1    #MARK                              A    1
     V   1    EMPLOYEES-VIEW                                   EMI
         2    PERSONNEL-ID                       A    8
         2    NAME                               A    20
         2    DEPT                               A    6
         2    LEAVE-DUE                          N    2.0
     *        *** End of Data Area ***
```

Step 3

The new data area must be stowed before any program referencing that data area can be compiled.

To stow the new data area

1. Stow GDA01 by choosing "Stow" from the "Object" menu.
2. Close GDA01 by choosing "Close" from the "Object" menu.

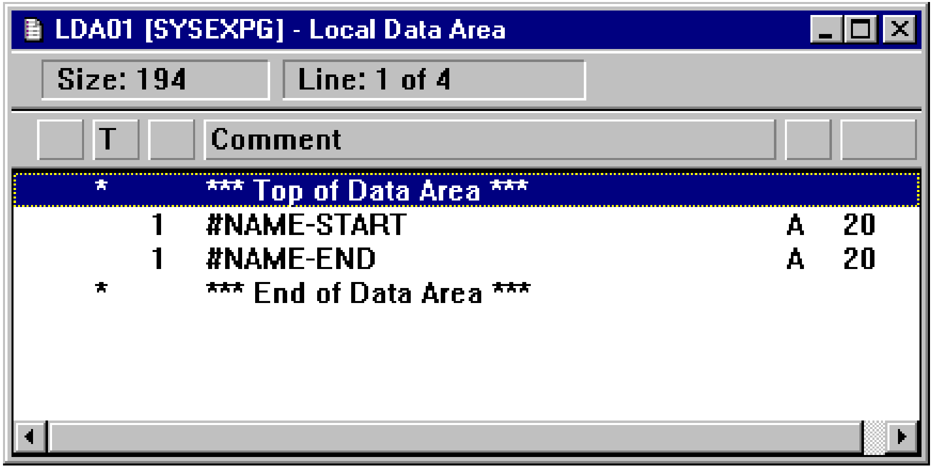
Step 4

Now that the new global data area has been created, the variables contained in it must be removed from the local data area.

Open LDA01.

To remove all the data fields that are now in the global data area GDA01 ("#MARK", "EMPLOYEES-VIEW", and all remaining lines)

1. Select all fields except "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.
   The revised local data area now contains only the variables "#NAME-START" and "#NAME-END":



```
*       *** Top of Data Area ***
    1   #NAME-START                    A   20
    1   #NAME-END                      A   20
*       *** End of Data Area ***
```

3. Stow the revised local data area.
   LDA01 is now ready to be referenced by the program PGM01.
4. Close LDA01.

Step 5

The DEFINE DATA statement in the PGM01 program must now reference data that are located in the global data area GDA01 as well as the local data area LDA01.

To open PGM01, and add a reference to the global data area

1. Open PGM01.
2. Place the cursor at the end of the DEFINE DATA statement and press ENTER.
3. In the blank line created, type GLOBAL USING GDA01 and press ENTER.

Step 6

In this step, you will revise the output instructions in PGM01.

In this step, you will modify the program PGM01 to include a WRITE TITLE statement, which produces a multiple-line title in the resulting report, and modify the format of the DISPLAY statement.

To do so

1. Insert a blank line after the following lines:
   ```
   RESET #MARK
   END-IF
   ```
2. Add the following WRITE TITLE statement:
   ```
   WRITE TITLE
     / ’*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***’
     / ’***     ARE MARKED WITH AN ASTERISK      ***’//
   ```
   The "/" notation indicates a line break. The title lines are centered and are not underlined.
3. Change the DISPLAY statement as follows:
   ```
   DISPLAY 23X ’//N A M E’  NAME
            3X ’//DEPT’     DEPT
            3X ’/LV/DUE’    LEAVE-DUE
            3X ’//*’         #MARK
   ```

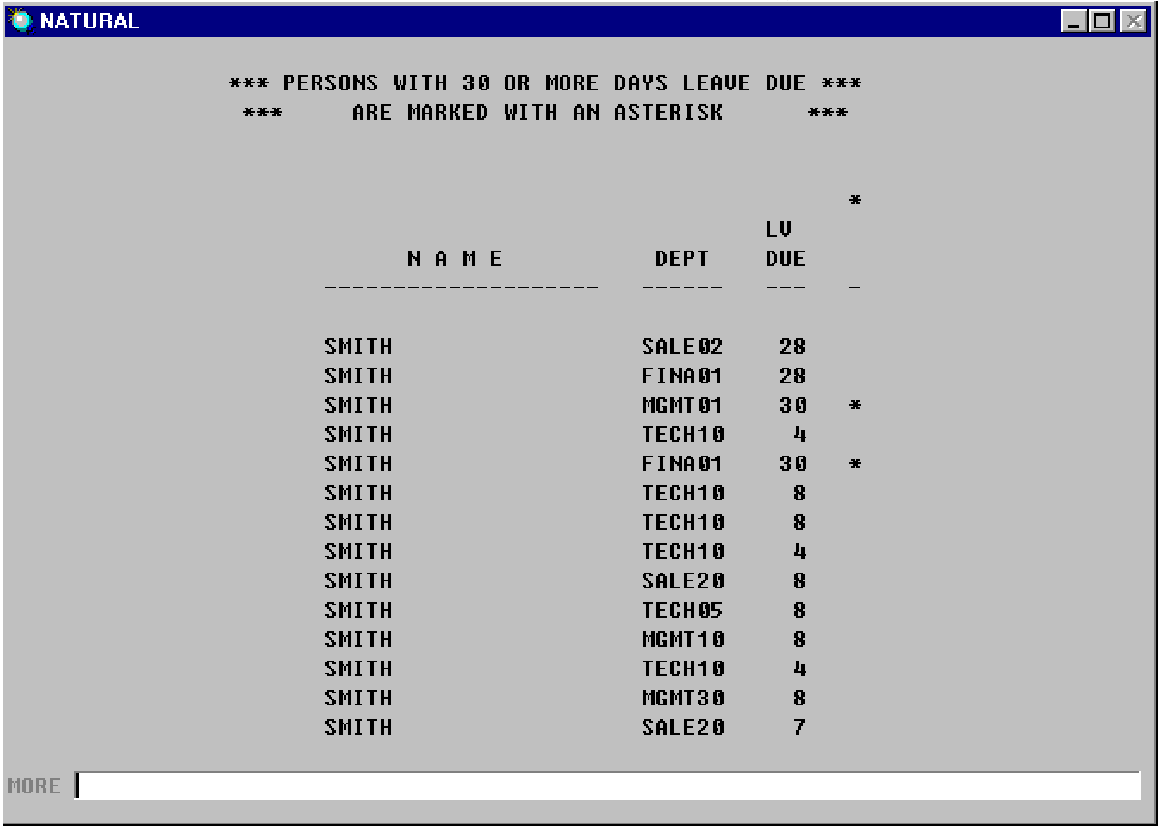
The revised program should now have the changes to the DEFINE DATA, WRITE TITLE, DISPLAY statements, and the program header (comment) as shown below.

```
* PGM-ID:   PGM01
* FUNCTION:   DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE DISPLAY STATEMENT HAS BEEN CHANGED
* -----------------------------------------------------
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL  USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP ’MAP01’
*
  IF #NAME = ’.’
    ESCAPE BOTTOM
  END-IF
*
  IF #END = ’ ’
    MOVE #NAME TO #END
  END-IF
*
  RD1.  READ EMPLOYEES-VIEW
          BY NAME
          STARTING FROM #NAME
          THRU #END
*
    IF LEAVE-DUE >= 30
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
  WRITE TITLE
       / ’*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***’
       / ’***     ARE MARKED WITH AN ASTERISK        ***’//
*
  DISPLAY 23X ’//N A M E’ NAME
           3X ’//DEPT’    DEPT
           3X ’/LV/DUE’   LEAVE-DUE
           3X ’//*’       #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT ’PLEASE TRY ANOTHER NAME’
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE ’*’ TO #MARK
END-SUBROUTINE
*
END
```

Step 7

After you have completed all changes:

1. Check the program and correct any errors that might exist.
2. Run the program, using "SMITH" as the name on the input screen.
   Note the differences in the report output, which should have the following format:

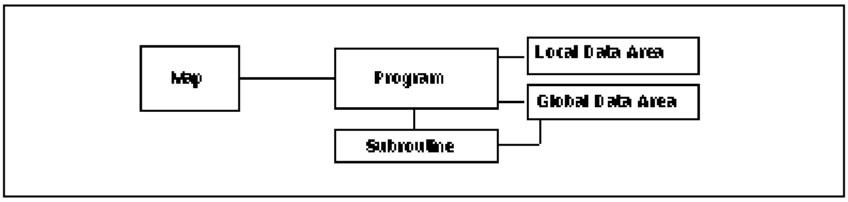


```
                *** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***
                 ***     ARE MARKED WITH AN ASTERISK      ***

                                                              *
                                                        LV
                             N A M E             DEPT   DUE
                       --------------------    ------   ---   -

                       SMITH                   SALE02    28
                       SMITH                   FINA01    28
                       SMITH                   MGMT01    30   *
                       SMITH                   TECH10     4
                       SMITH                   FINA01    30   *
                       SMITH                   TECH10     8
                       SMITH                   TECH10     8
                       SMITH                   TECH10     4
                       SMITH                   SALE20     8
                       SMITH                   TECH05     8
                       SMITH                   MGMT10     8
                       SMITH                   TECH10     4
                       SMITH                   MGMT30     8
                       SMITH                   SALE20     7

MORE
```

3. After you have confirmed that PGM01 has no errors, stow it for future modification in Session 6 and close PGM01.

End of Session 5.

Session 6 - Creating an External Subroutine



In Natural, a subroutine can be defined either within a program, or as an external subroutine outside the program.

Until now, the subroutine "MARK-SPECIAL-EMPLOYEES" has been defined within the program using a DEFINE SUBROUTINE statement. In this session, the subroutine will be defined as a separate object external to the program.

Because both internal and external subroutines are invoked with a PERFORM statement, only minimal changes to the program are required.

Step 1

In this step, you will create a subroutine named SUBR01:

Note:
This subprogram is contained in library SYSEXPG. If you have access to this library, you do not have to perform this step.

To open a new program editor window

1. From the **Object** menu, choose **New.**
2. From the cascading menu, choose **Subroutine**.
3. Enter the following statements:

```
* SUBR-ID:  SUBR01
*
* FUNCTION:  DEMONSTRATE NATURAL
* THIS IS A SUBROUTINE
*
*
*
*
*---------------------------------------------------
```

To save the subroutine

1. From the **Object** menu, choose **Save As**.
   The "Save As" dialog box appears.
2. In the "Name" text box, enter "SUBR01".
   SUBR01 should be saved in the SYSEXPG library. If SYSEXPG is not the current library, from the "Library" list box, select SYSEXPG.
3. Choose **OK**.

Step 2

In this step, you will edit the program PGM01, copy two statements and paste them into the subroutine SUBR01.

To edit the program PGM01

1. Use the **Minimize** button to minimize SUBR01.
   Note:
   You can reopen SUBR01 by clicking its icon or by choosing "SUBR01" from the "Window" menu.
2. Open PGM01.

To copy the DEFINE DATA statement

1. Place the cursor at the beginning of the DEFINE DATA statement and drag the mouse until the following lines are selected:
   ```
   DEFINE DATA
     GLOBAL USING GDA01
     LOCAL  USING LDA01
   END-DEFINE
   *
   ```
2. From the **Edit** menu, choose **Copy**.
   The DEFINE DATA statement is copied and placed on the clipboard.
3. Use the **Minimize** button to minimize PGM01.

To paste the copied statement into SUBR01

1. From the **Window** menu, choose **SUBR01**.
2. Place the cursor below the last comment line.
3. From the **Edit** menu, choose **Paste.**
   The DEFINE DATA statement appears.
4. Cut the following DEFINE SUBROUTINE block from PGM01 and paste it into SUBR01. Follow the same procedure as above but, from the **Edit** menu, choose **Cut** instead of **Copy.**
   ```
   DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
     MOVE '*' TO #MARK
   END-SUBROUTINE
   *
   ```
5. Paste the block below the END-DEFINE in program PGM01.
6. Add an END statement at the end of the subroutine.

Step 3

The subroutine SUBR01 should now appear as follows:

```
* SUBR-ID:  SUBR01
*
* FUNCTION: DEMONSTRATE NATURAL
* THIS IS A SUBROUTINE
*
*
*
*
* -----------------------------------------------------
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
    MOVE ’*’ TO #MARK
END-SUBROUTINE
*
END
```

1. From the **Object** menu, choose **Check** to check SUBR01 and correct any errors.
2. From the **Object** menu, choose **Stow** to stow SUBR01.
3. From the **Object** menu, choose **Close** to close SUBR01.

The program PGM01 should now look as follows:

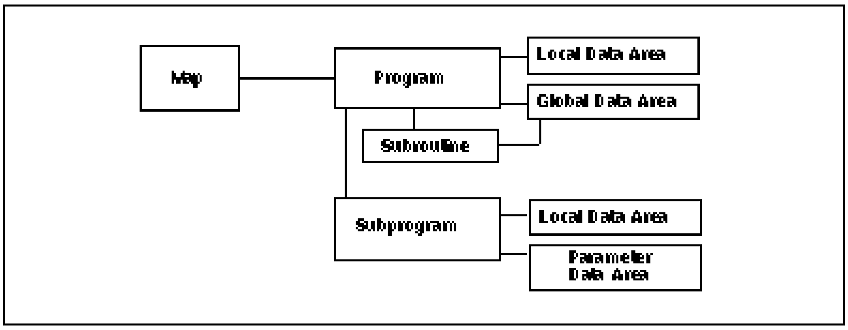
```
* PGM-ID:   PGM01
* FUNCTION:   DEMONSTRATE NATURAL
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE DISPLAY STATEMENT HAS BEEN CHANGED
* THE SUBROUTINE IS NOW EXTERNAL
* -----------------------------------------------------
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL  USING LDA01
END-DEFINE
*
REPEAT
*
    INPUT USING MAP ’MAP01’
*
  IF #NAME-START = ’.’
    ESCAPE BOTTOM
  END-IF
*
  IF #NAME-END = ’ ’
    MOVE #NAME TO #NAME-END
  END-IF
*
  RD1.  READ EMPLOYEES-VIEW
          BY NAME
          STARTING FROM #NAME-START
          THRU #NAME-END
*
    IF LEAVE-DUE >= 30
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
  WRITE TITLE
      / ’*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***’
      / ’***     ARE MARKED WITH AN ASTERISK        ***’ //
*
  DISPLAY 23X ’//N A M E’ NAME
           3X ’//DEPT’    DEPT
           3X ’/LV/DUE’   LEAVE-DUE
           3X ’//*’       #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT ’PLEASE TRY ANOTHER NAME’
  END-IF
*
END-REPEAT
*
END
```

Step 4

1. Check PGM01 and correct any errors.
2. Run the program to confirm that the results are the same with an external subroutine as with an internal subroutine.
3. Stow the program for the next session.
4. Close PGM01, saving your changes.

End of Session 6.

Session 7 - Invoking a Subprogram



In Natural, both subprograms and subroutines can be invoked from a main program.

A subprogram is invoked using a CALLNAT statement. Data are passed from the main program (the calling program) to a subprogram through a set of parameters that are referenced or defined in the DEFINE DATA PARAMETER statement of the subprogram.

While a subroutine such as SUBR01 created in Session 6 shares a global data area with the main program, the subprogram only receives data that are passed by way of a parameter list from the main program's CALLNAT statement.

In this session, the PGM01 program will be expanded to include a CALLNAT statement that invokes a subprogram. In the subprogram, the employees identified from the main program will be the basis of a FIND request to the VEHICLES file. As a result, your report will contain VEHICLES information from the subprogram as well as leave due, etc. from the main program.

The new subprogram will require the creation of a local data area and a parameter data area. In this case, new variables will be defined in the main program's local data area, and this will in turn help create the subprogram's parameter data area variables.

Step 1

The local data area that you created in Session 4 (LDA01) is stored in the SYSEXPG library. Make sure that the SYSEXPG library is the current library.

In this step, you will modify the LDA01 local data area to accommodate the new subprogram. The following fields must be added to LDA01:

```
   #PERS-ID
   #MAKE
   #MODEL
```

These fields are referenced in the CALLNAT statement that you will add to the program PGM01 in a later step.

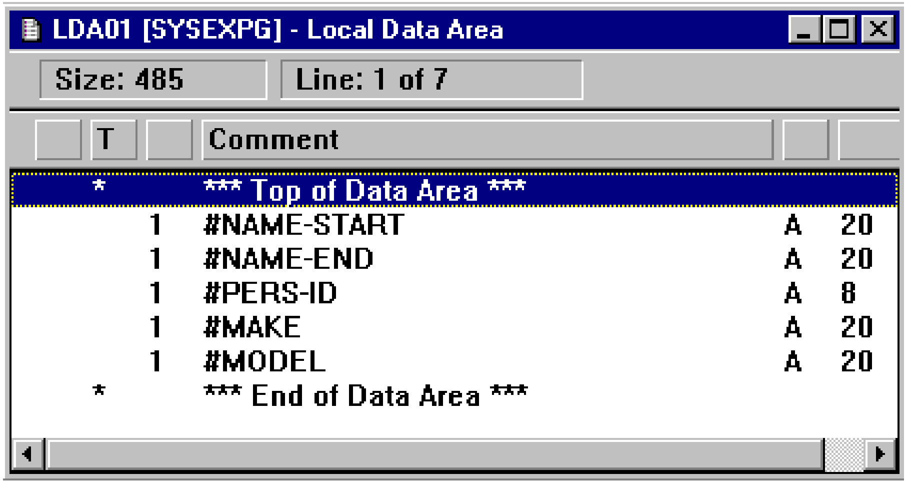
Open LDA01.

To add the data fields

1. Select the field "#NAME-END".
2. From the **Insert** menu, choose **Data Field**.
   The "Data Field Definition" dialog box appears.
   In the "Level" text box, the default "1"is displayed.
3. In the "Name" text box, enter "#PERS-ID".
4. In the "Length" text box, enter "8".
5. Choose **Add**.
   The field definition you entered is added to the LDA01 local data area, and the "Data Field Definition" dialog box reappears, allowing you to define the next data field.

To define the two remaining fields, "#MAKE" and "#MODEL", as you defined the "#PERS-ID" field, enter a length of "20" for each field

1. Choose **Quit** to close the "Data Field Definition" dialog box and return to the data area editor window.
   The local data area should now appear as follows.



```
Size: 485      Line: 1 of 7

   T      Comment
*         *** Top of Data Area ***
     1    #NAME-START                        A   20
     1    #NAME-END                          A   20
     1    #PERS-ID                           A   8
     1    #MAKE                              A   20
     1    #MODEL                             A   20
*         *** End of Data Area ***
```

2. Check and stow the LDA01 local data area.

Step 2

With minor modifications, the LDA01 local data area can be used to create the parameter data area that will be needed for the subprogram.

In this step, you will delete two of the data fields in LDA01, then save the revised data area as a parameter data area named PDA01. The original LDA01 local data area remains intact. (It is also possible to define the parameter data area directly by using the menu to choose "Object > New > Parameter data area").

Open LDA01.

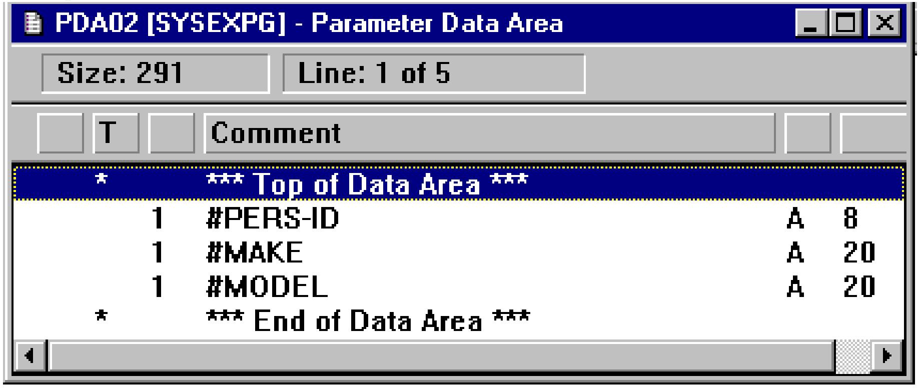
To delete the data fields "#NAME-START" and "#NAME-END"

1. Select the fields "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.

To save the data area with the name PDA02 and data area type "Parameter"

1. From the **Object** menu, choose **Save As**.
   The "Save As" dialog box appears.
2. In the "Name" text box, enter "PDA02".
3. Open the "Type" list box and select "Parameter".

4. Choose **OK.**
   Your parameter data area should now look as follows.



```
   *      *** Top of Data Area ***
      1   #PERS-ID                                  A    8
      1   #MAKE                                     A   20
      1   #MODEL                                    A   20
   *      *** End of Data Area ***
```

5. Check the new parameter data area and correct any errors.
6. In the SYSEXPG library, stow the parameter data area.
7. Close the parameter data area.

Step 3

The subprogram will also use variables that are local to the program. In this step, you will create a new local data area.

To open a new data area window to create a local data area

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Local data area**.

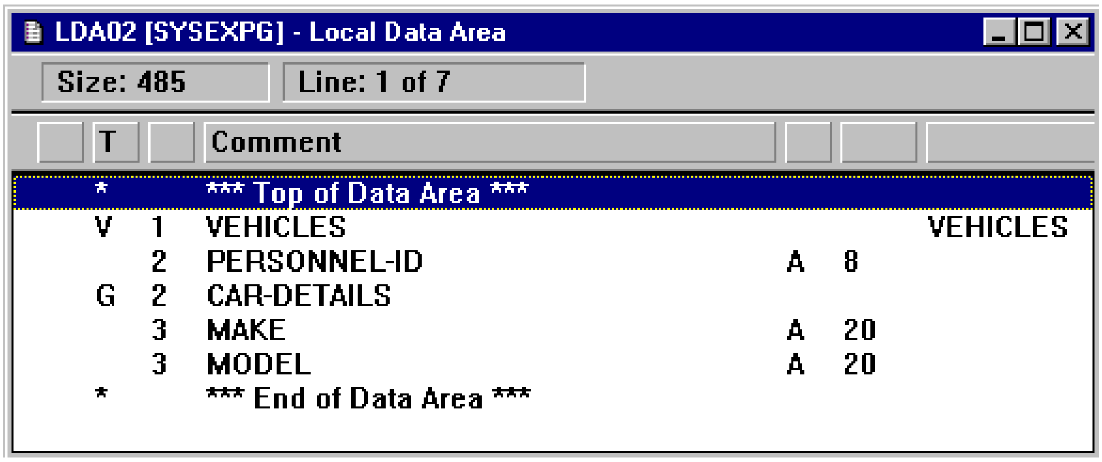
Step 4

Fields contained in any Natural DDM can be imported into a data area. In this step, you will import several fields from the VEHICLES DDM into the new local data area.

To import fields from the VEHICLES DDM

1. From the **Insert** menu, choose **Import**.
   The "Import View" dialog box appears with the name of the current library (SYSEXPG) in the "Library" list box.
2. Open the "Library" list box and select the SYSEXDDM library.
   A list of all DDMs in the SYSEXDDM library appears in the DDM list box.
3. Select the "VEHICLES" DDM.
   A list of all the data fields in the "VEHICLES" DDM appears in the "Data fields" list box.
4. Select the fields "PERSONNEL-ID" through "MODEL" (drag the mouse across the fields to select them) and choose OK.
5. In the "View Definition" dialog, choose **OK**.
   The fields appear in the data area window.

The local data area now contains fields imported from the "VEHICLES" DDM as shown below:



```
*         *** Top of Data Area ***
V    1    VEHICLES                                    VEHICLES
     2    PERSONNEL-ID                      A   8
G    2    CAR-DETAILS
     3    MAKE                              A   20
     3    MODEL                             A   20
*         *** End of Data Area ***
```

Step 5

▶ To save the new local data area as LDA02

1. From the **Object** menu, choose **Save As**.
   The "Save As" dialog box appears.
2. In the "Name" text box, enter "LDA02".
3. Choose **OK**.
   The local data area is saved as LDA02 in the SYSEXPG library.
4. Check the new local data area and correct any errors.
5. Stow the new local data area.
   LDA02 is now ready for use by the subprogram.
6. Close LDA02.

Step 6

The subprogram used in this session, SPGM02, receives the personnel number passed by the main program (PGM01) and uses this number as the basis for a search of the VEHICLES file.

The SYSEXPG demo library should include the SPGM02 subprogram.

If SPGM02 is available, ensure that it has been stowed and then proceed directly to Step 7 (modifying the main program) later in this session.

If SPGM02 is not available, you can create it. Instructions are provided below.

To open a new program editor window to create the subprogram

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Subprogram**.
3. Enter the subprogram shown below:

```
* PGM-ID:  SPGM02
* ----------------------------------------------------
DEFINE DATA
  PARAMETER
    USING PDA02
  LOCAL
    USING LDA02
END-DEFINE
*
FD1.  FIND (1) VEHICLES
          WITH PERSONNEL-ID = #PERS-ID
        MOVE MAKE (FD1.)    TO   #MAKE
        MOVE MODEL (FD1.)   TO  #MODEL
        ESCAPE BOTTOM
END-FIND
*
END
```

4. Save SPGM02 and stow it.
5. Close SPGM02.

Step 7

In this step, you will modify the main program (PGM01) to accommodate the subprogram.

To do so

1. Open PGM01.
2. Add the following statements immediately before the WRITE TITLE statement:

```
RESET #MAKE #MODEL
CALLNAT 'SPGM02' PERSONNEL-ID #MAKE #MODEL
```

The parameters passed in the CALLNAT statement come from both the global data area and the local data area. Also, the variables defined in the parameter data area of the subprogram do not have to have the same name as the variables in the CALLNAT statement. Because the parameters are passed by address, it is only necessary that they match in sequence, format, and length.

Because the subprogram is now returning vehicle information, the DISPLAY statement must be modified as shown below:

```
  *
    WRITE TITLE
        / ′*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***′
        / ′***     ARE MARKED WITH AN ASTERISK        ***′ //
    DISPLAY  1X ′//N A M E′ NAME
             1X ′//DEPT′    DEPT
             1X ′/LV/DUE′   LEAVE-DUE
                ′ ′         #MARK
             1X ′//MAKE′    #MAKE
             1X ′//MODEL′   #MODEL
```

1. Check PGM01 and correct any errors.
2. Run PGM01.
3. Stow PGM01. Close PGM01.

End of Session 7.

Class Builder

The following topics are covered below:

- Introduction
- What is the Class Builder?
- Class Builder Interface
- Class Builder Nodes
- Node Properties
- Adding Class Components
- Renaming Class Components
- Removing Class Components
- Editing Class Components
- Using Interfaces from several Classes
- Locking Concept
- Tutorial
- Glossary

Introduction

The Class Builder is a tool which can be used to display a Natural class in a structured hierarchical order, and also to manage the class and its components efficiently.

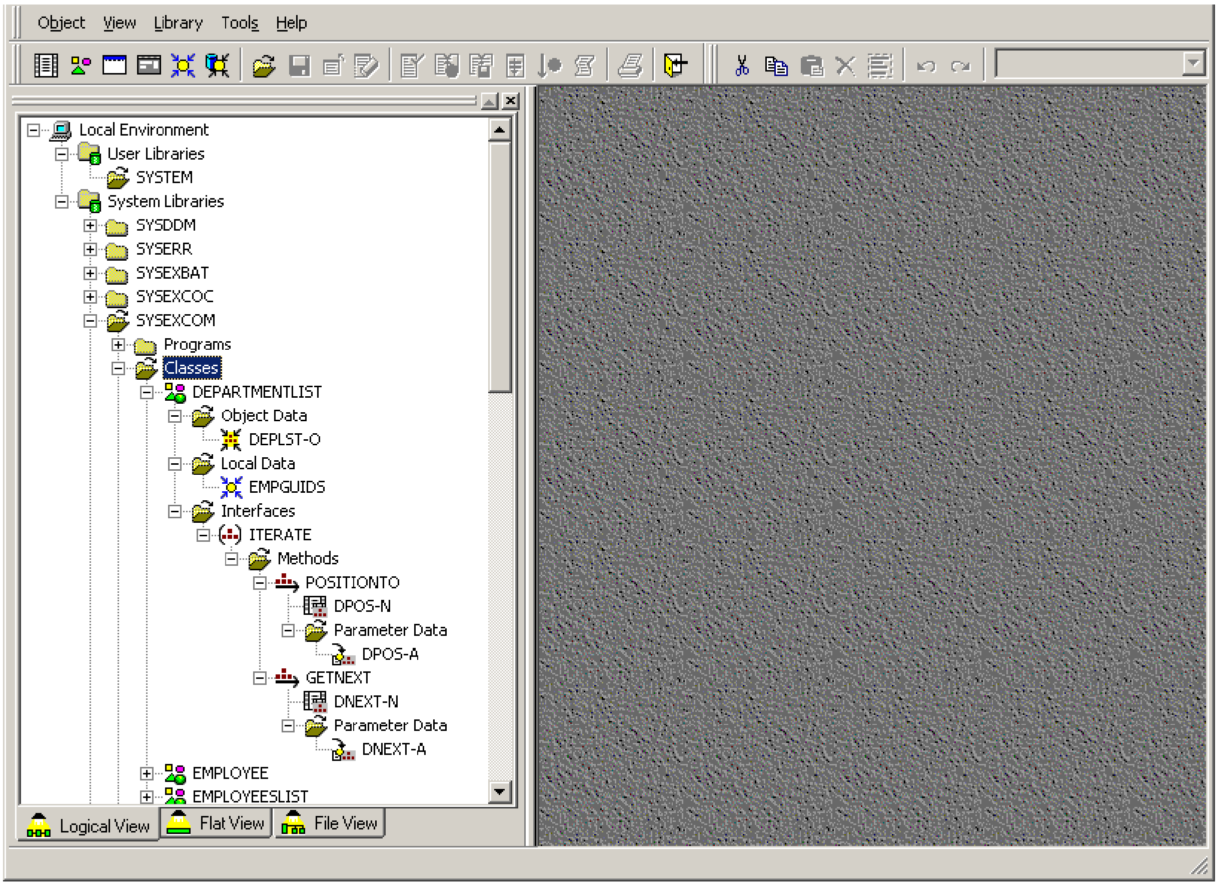
A Natural class can be composed of various components: "real" Natural objects (for example, an object data area) or objects which exist only in the class source (for example, interface components).
The Class Builder represents each component of the class in the form of a node. By selecting these nodes, the class and its components can be managed in a context-sensitive manner.

This section explains how to create and modify a Natural class with the Class Builder. Please refer to the NaturalX documentation to become acquainted with the general usage of Natural classes.

What is the Class Builder?

The Class Builder provides the following features:

- It is fully integrated in the general Natural user interface.
- The components of a class are displayed as nodes (library workspace or list view) in the same way as Natural modules. Every type of node has a special icon assigned which provides detailed information for that component.
- Natural objects which are used by a class (for example, ODA), can be managed (edit, stow, ...) by the Class Builder.
- Class and interface GUIDs (Global Unique IDs) are generated and hidden.
- Class comments (one comment for every class component) can be created and changed by the Class Builder.
- The class source is generated automatically.



Which Classes can be handled by the Class Builder?

The Class Builder can manage any class which can be successfully checked. No special statements must be written in the class source. This means that it is possible to change classes which have been generated with the Class Builder with the program editor and vice versa. This is especially important if a class is used on several platforms in that the Class Builder is not available on all Natural platforms.

The class syntax is highly "flexible", i.e., it is possible to obtain the same runtime behavior with different syntax constructs. This was important for earlier Natural versions, because the user had to type all class code himself. With the Class Builder, this is no longer necessary; the Class Builder will generate the class code and create Natural objects, which are used by the class. The Class Builder will generate only the most reasonable code. For this reason, the following features are not supported by the Class Builder:

- **create a new GUID LDA**: The Class Builder generates a GUID for the class and the interfaces of the class. If you wish to define the GUID yourself, you must create a LDA outside of the Class Builder and then link it to the class.
- **create new inline data definitions**: The Class Builder only provides for the creation of new data areas. This is because data definitions are usually used in several places (for example, method parameter in class and method subprogram) and it is fault-prone if the same inline data definitions have to be used more than once.
- **use data from inline data definitions for assignments in the Class Builder**: If data definitions have to be assigned to class components in the case of unique IDs and property implementations, the Class Builder offers a list of all data definitions from the corresponding data areas. Data from inline data definitions will not be included in these lists. This means, for example, that the object data variable which is defined inline cannot be used as property implementation.

Although the Class Builder does not permit the creation of all class syntax constructs, it can nonetheless read existing classes with these constructs and can be used to modify these constructs.

If the Class Builder cannot read a class because it is syntactically incorrect, it displays an error message and activates the program editor. The syntax error must be corrected in the program editor. After the class has been saved, it can be opened with the Class Builder.

Note: If you save a class with the Class Builder, the class source will be generated. This means that any special source formats, such as indentation, will be lost.

When is a Class saved?

When a class is opened in the Class Builder, the contents are read from the class source and stored in an internal structure. If you then change the class, these changes are performed only on the internal structures. The changes are visible in all views of Natural. So, for example, when a new interface is added in the library workspace, a node for this interface will also be created in the "Interfaces" list view of the class. If you want to save your changes, you must execute "Save", "Save As" or "Stow" for the class.

If you create a new class, this does not automatically create a new class module. This is only done when "Save", "Save As" or "Stow" is executed for the class. For this reason, a "new" class will not be visible in the File View of the library workspace until it is saved the first time.

If you want to remove the changes which you applied to a class, you can use the "Restore" command. This command will restore the class as it is contained in the class module, i.e., the last saved state.

If Natural is ended and unsaved classes exits, the user will be asked if the classes should be saved.

Class Comments

The Class Builder tries to assign every comment found in the class source to one component of the class. A comment is usually assigned to the following class component. For example a comment which is found before the definition of an interface is taken as comment for this interface.

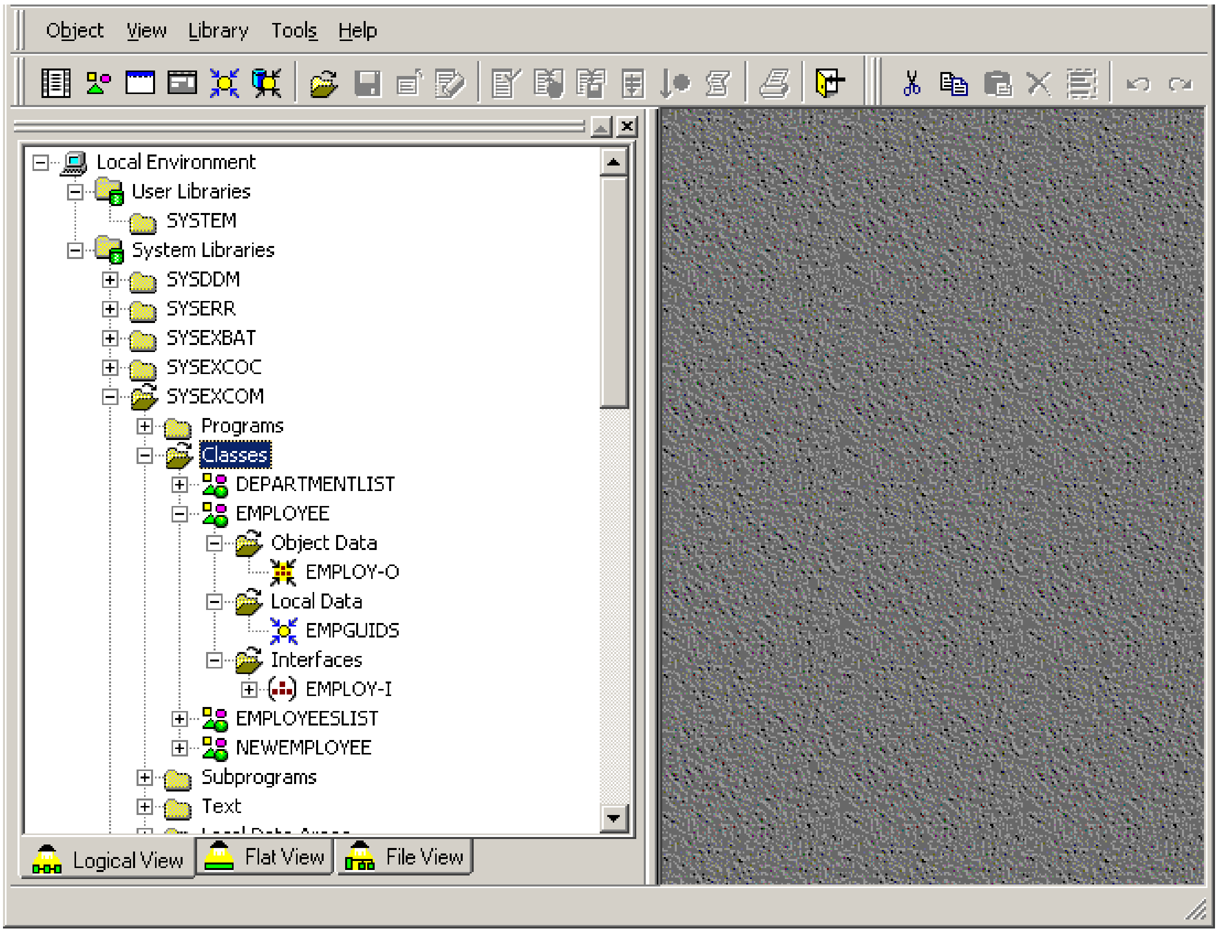
The comments can be changed and created via the "Properties" menu item, which is available for all class component nodes. For more information, see Node Properties.

Note: If a class is read by the Class Builder for the first time, it is possible that the Class Builder assigns the comments to a component other than the one the user expects. No comment will be lost when the class is saved, but the user should check if the comments are assigned to the correct components.

When a class is saved by the Class Builder for the first time, all comments will be marked with a special tag. This ensures that the comment assignment is correct when this class is read later by the Class Builder.

Class Builder Interface

The Class Builder is available in the logical and flat view of Natural. It is fully integrated in the general Natural user interface which shows the Natural objects as nodes of a tree or list view.
In the Library Workspace, a class can be "opened" by expanding the class node. The class nodes are grouped hierarchically. For example, the interface is a child of the class node and the method is a child of the interface node. Every class node provides the same features as all other nodes, for example, a context menu which allows node-specific actions. Most of the class nodes that have child nodes can be opened as a list view which displays all children of this node. The List View shows some more information about the nodes (for example, the library in which an object data area is located). The list view nodes offer the same context menu as the corresponding library workspace nodes. The columns of a list view can be sorted alphabetically.



Logical View

The class nodes of the logical view are inscribed with the class name, i.e., the name that is used when an object of this class is created with the CREATE OBJECT statement.
In the logical view the nodes are, as a basic principle, grouped by their type. This is also valid for the class nodes. Class nodes of the same type are collected under a group node which describes the type with its contents. Therefore, all object data nodes are children of the object data group node named "Object Data".

Library Workspace

You can expand and collapse nodes of a class in the Library Workspace. Expand displays all child nodes and Collapse hides all child nodes of the selected class node.
The logical view provides you with a structured view of the class. You can then expand those class nodes on which you wish to work. For more information, refer to the Natural Studio documentation.

List Views

Most of the parent nodes of a class have an assigned list view which can be opened with the "Open" command from the context menu. This section describes the information which is shown in the list views of the logical view.For more information about List Views, refer to the Natural Studio documentation.

Class List View

The class list view consists of group nodes. The list view for a group node can be opened with the "Open" command.

The following group nodes exist:

- **"Object Data" group**: is displayed if the class uses a ODA
- **"Local Data" group**: is displayed if the class uses a LDA for class or interface GUIDs
- **"Interface Modules" group**: is displayed if the class uses an Interface Module (see Using Interfaces from several Classes).
- **"Interfaces" group**: is displayed if the class has defined interfaces (internal or external)

The class list view has the following columns:

- **Type**: type of the node (e.g. Object Data)
- **Count**: number of components of the specified type

Object Data Group List View

The "Object Data" group list view consists of object data nodes. Choosing the "Open" command for a node will open the data area editor for data areas and a special Class Builder dialog for inline definitions.

The "Object Data" group list view has the following columns:

- **Name**: name of the object data module or "Inline" in the case of an inline data definition
- **Library**: library where the object data module is located (is empty for inline data definitions or if the data area has not yet been created)
- **Type**: Natural type of the object data module ("Local Data Area", "Parameter Data Area" or "Inline Definition")

Local Data Group List View

The "Local Data" group list view consists of local data nodes. Choosing the "Open" command for a node will open the data area editor for data areas and a special Class Builder dialog for inline definitions.

The "Local Data" group list view has the following columns:

- **Name**: name of the local data module or "Inline" for an inline data definition.
- **Library**: library where the local data module is located (empty for inline data definitions or if the data area has not yet been created).
- **Type**: Natural type of the local data module ("Local Data Area", "Parameter Data Area" or "Inline Definition").

Interface Modules Group List View

The "Interface Modules" group list view consists of interface module nodes (see Interface Module List View). Choosing the "Open" command for a node will open the list view (see Using Interfaces from several Classes) for this particular interface module.

The "Interface Modules" group list view has the following columns:

- **Name**: name of the interface module (copycode name)
- **Library**: library where interface module is located.

Interface Module List View

The interface module list view consists of interface nodes. Choosing the "Open" command for a node will open the list view (see Interface List View) for this particular interface.

The interface module list view has the following columns:

- **Name**: name of the interface.

Interfaces Group List View

The "Interfaces" group list view consists of interface nodes. Choosing the "Open" command for a node will open the list view (see Interface List View) for this particular interface.

The "Interfaces" group list view has the following columns:

- **Name**: name of the interface.
- **Component Type**: "Internal Interface" for interfaces which are defined in the class and "External Interface" for interfaces which are defined in an interface module included in this class.
- **Defined In**: interface module name for externally defined interfaces (empty for internal interfaces).

Interface List View

The interface list view consists of group nodes. Choosing the "Open" command for a node will open a list view for this particular group.

The following group nodes exist:

- **"Properties" group**: is displayed if the interface contains property definitions
- **"Methods" group**: is displayed if the interface contains method definitions.

The interface list view has the following columns:

- **Type**: type of the node (e.g. Properties).
- **Count**: number of components of the specified type.

Properties Group List View

The "Properties" group list view consists of property nodes. The "Property" group list view has the following columns:

- **Name**: name of the property.
- **Format**: format of property.
- **Length**: length of property.
- **Dimension**: dimension of property.
- **Read-only**: shows whether property is read-only or not.
- **ODA Variable**: name of assigned ODA variable

Methods Group List View

The "Methods" group list view consists of method implementation and parameter data nodes. For every method of the interface, it contains one method implementation (subprogram) node and one node for every parameter data definition of the method.
Choosing the "Open" command for a node of this list view will open the editor for the particular node type (for example, program editor for method implementation node).

The "Methods" group list view has the following columns:

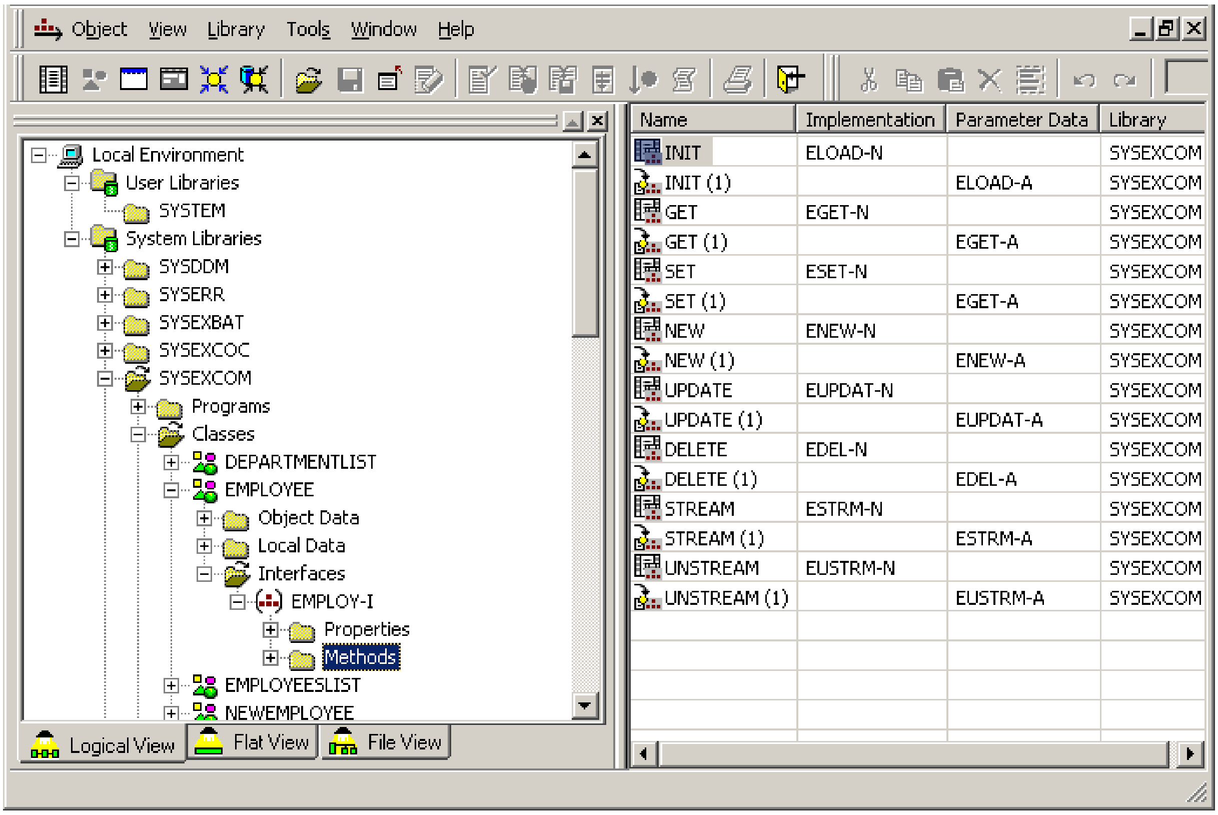
- **Name**: name of the method. The parameter data nodes are numbered from 1 to n (for example, INIT (2) for the second parameter data node of method INIT).
- **Implementation**: only for method implementation node: the name of the subprogram which implements the method
- **Parameter Data**: only for method parameter data node: the name of the parameter data module or "Inline" for an inline data definition
- **Library**: depending on the node type, library where implementation or parameter data module is located (empty for inline data definitions or if the Natural module has not yet been created).

Method Parameter Data Group List View

The "Parameter Data" group list view consists of parameter data nodes. Choosing the "Open" command for a node will open the data area editor for data areas and a special Class Builder dialog for inline definitions.

The "Parameter Data" group list view has the following columns:

- **Name**: name of the parameter data module or "Inline" for an inline data definition.
- **Library**: library where parameter data module is located (empty for inline data definitions or if the data area has not yet been created)
- **Type**: Natural type of parameter data module ("Parameter Data Area" or "Inline Definition")



Name	Implementation	Parameter Data	Library
INIT	ELOAD-N		SYSEXCOM
INIT (1)		ELOAD-A	SYSEXCOM
GET	EGET-N		SYSEXCOM
GET (1)		EGET-A	SYSEXCOM
SET	ESET-N		SYSEXCOM
SET (1)		EGET-A	SYSEXCOM
NEW	ENEW-N		SYSEXCOM
NEW (1)		ENEW-A	SYSEXCOM
UPDATE	EUPDAT-N		SYSEXCOM
UPDATE (1)		EUPDAT-A	SYSEXCOM
DELETE	EDEL-N		SYSEXCOM
DELETE (1)		EDEL-A	SYSEXCOM
STREAM	ESTRM-N		SYSEXCOM
STREAM (1)		ESTRM-A	SYSEXCOM
UNSTREAM	EUSTRM-N		SYSEXCOM
UNSTREAM (1)		EUSTRM-A	SYSEXCOM

Flat View

The class nodes of the flat view show the class module name.
Unlike the logical view, the flat view does not contain any group nodes. The flat view has the advantage that the level where a specific class component is displayed is lower compared to the logical view, and thereby provides you with a better class overview.

Library Workspace

You can expand and collapse nodes of a class in the Library Workspace. Expand displays all child nodes and Collapse hides all child nodes of the selected class node. The flat view provides you with a general overview of the class. It lists all sub-components of a class component on the same level. For example, if an interface node is expanded, all properties and methods of the interface will be displayed as child nodes of the interface node. For more information, see Natural Application Development Environment.

List Views

The flat view supports only a few list views because of the low node nesting level. The list views can be opened with the "Open" command from the context menu. This section describes the information which is shown in the list views of the flat view. For more information about List Views, refer to the Natural Studio documentation.

Class List View

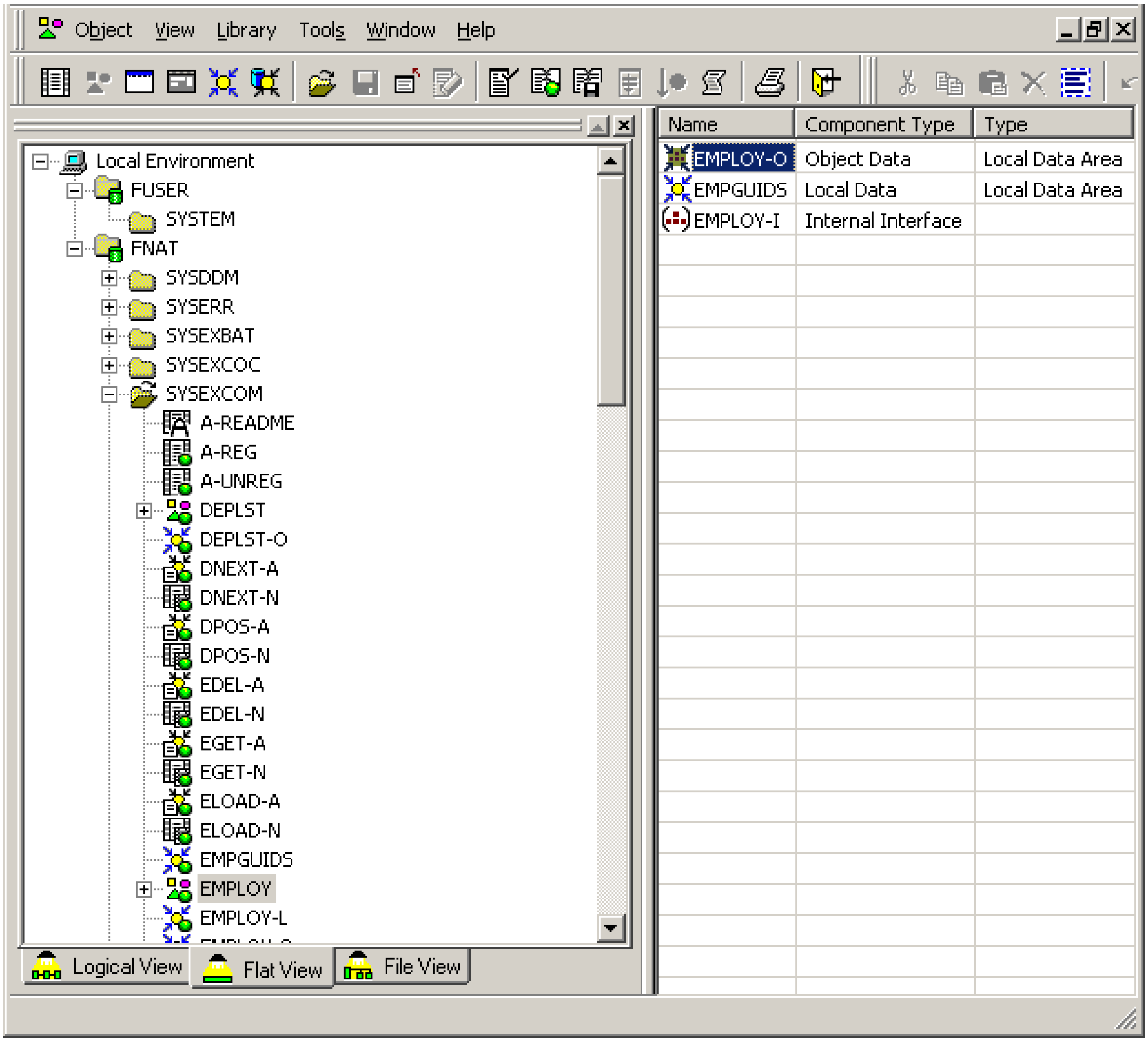
The class list view contains a node for every child component.

The following nodes exist:

- **Object Data node** for every ODA of the class. Choosing the "Open" command of the node opens the data area editor for data areas and a special Class Builder dialog for inline definitions
- **Local Data node** for every GUID LDA of the class. Choosing the "Open" command of the node opens the data area editor for data areas and a special Class Builder dialog for inline definitions.
- **Interface Module node** for every interface module which is used by the class.Choosing the "Open" command of the node will open the interface module list view.
- **Interface node** for every interface of the class (external and internal). Choosing the "Open" command of the node will open the interface list view.

The class list view has the following columns:

- **Name**: name of the component.
- **Component Type**: indicates the type of the component ("Object Data", "Local Data", "Interface Module", "External Interface" or "Internal Interface").
- **Type**: only for component type "Object Data" and "Local Data": Natural type of data module ("Local Data Area", "Parameter Data Area" or "Inline Definition")



Name	Component Type	Type
EMPLOY-O	Object Data	Local Data Area
EMPGUIDS	Local Data	Local Data Area
EMPLOY-I	Internal Interface	

Interface Module List View

The interface module list view consists of interface nodes. Choosing the "Open" command of a node will open the list view (see List View) for this particular interface.

The interface module list view has the following columns:

- **Name**: name of the interface.

Interface List View

The interface list view contains all nodes for the properties and methods of the interface.

The following nodes exist:

- **Property node** for every property of the interface.
- **Method implementation node** for every method of the interface. Choosing the "Open" command for the node will open the program editor with the specified implementation (subprogram).
- **Method parameter data node** for every parameter data component of every method of the interface. Choosing the "Open" command for the node will open the data area editor for data areas and a special Class Builder dialog for inline definitions.

The interface list view has the following columns:

- **Name**: name of the property or method; the parameter data nodes for methods are numbered from 1 to n (for example, INIT (2) for the second parameter data node of method INIT).
- **Implementation**: only for properties and method implementation node: the name of the assigned ODA variable for properties and the name of the subprogram which implements the method for methods.
- **Parameter Data**: only for method parameter data node: the name of the parameter data module or "Inline" for an inline data definition.
- **Library**: only for methods: depending on the node type, library where implementation or parameter data module is located (empty for inline data definitions or if the Natural module has not yet been created).
- **Format**: only for properties: format of property.
- **Length**: only for properties: length of property.
- **Dimension**: only for properties: dimension of property.
- **Read-only**: only for properties: shows whether property is read-only or not.

Class Builder Nodes

Related to the user interface, every component of a class is represented by a node. Nodes are displayed both in the library workspace and in the list views.
Every node has an icon and textual information about the component which can be the name of the component (in the library workspace) or the name of the component and additional information (in the list views).

The following table lists all available Class Builder nodes with their icons and a short description:

Type	Icon	Description
new class		new class which has not yet been saved
class (src)		class which is only available as source
class (gp)		class which is only available as generated program
class (src & gp)		class which is available as source and generated program
ODA		object data defined in a data area module
inline ODA		object data defined with an inline data definition
LDA		local data (for GUIDs) defined in a data area module
inline LDA		local data (for GUIDs) defined with an inline data definition
Interface Module		interface module, i.e., copycode which defines interfaces
internal interface		interface which is defined in the class
external interface		interface which is defined in an interface module that is used by the class
internal property		property which is defined in an internal interface
external property		property which is defined in an external interface
internal method		method which is defined in an internal interface
external method		method which is defined in an external interface
method implementation		subprogram which implements a method
method PDA		method parameter data defined in a data area module
inline method PDA		method parameter data defined with an inline data definition

In the following section, the Class Builder nodes are described in more detail. The commands of a specific node can be invoked from the context menu of the node or the "Classes" toolbar.

Class Nodes

The class node represents the class itself. The name displayed in the class node is either the class name (logical view) or the class module name (flat view).

Types

New Class

If a new class is created, it is displayed with the new class icon until it is saved the first time. Therefore, new class means that the class is only "transient" in the current Natural session and is not available in source format. For this reason, the new class will not be shown in the File View which shows the source and gp files of the Natural objects. In addition, it is not possible to execute all class node commands on a new class.

Source-Only

The source-only class icon is displayed if the class is only available in source format but has not yet been cataloged.

GP-only (icon)

The GP-only class icon is displayed if the class is only available in GP format. Classes of this type cannot be handled with the Class Builder and the context menu of these classes is the same as for all other Natural objects which are only available in GP format.

Source-and-GP

The Source-and-GP class icon is displayed if the class is available in source and GP format.

Commands

Command	available for	Description
Open	new source-only source-and-GP	Opens the class list view. For more information, see List Views
List	new source-only source-and-GP	Opens the program editor in read-only state with the internal source format of the current class structure.
Cat	source-only source-and-GP	Catalogs the current class.
Save	new source-only source-and-GP	Saves the current class structure in the given class module.
Save As	new source-only source-and-GP	Saves the current class structure in a new Natural module.
Stow	new source-only source-and-GP	Stows the current class structure in the given class module.
New ODA	new source-only source-and-GP	Creates a new object data area for the class.

Command	available for	Description
New Interface	new source-only source-and-GP	Creates a new interface for the class.
New Interface Module	new source-only source-and-GP	Creates a new interface module. This interface module is linked to the class.
Link LDA	new source-only source-and-GP	Uses an existing data area as GUID LDA for the class. See Link.
Link ODA	new source-only source-and-GP	Uses an existing data area as ODA for the class. See Link.
Link Interface Module	new source-only source-and-GP	Uses an existing copycode as interface module for the class. All interfaces defined in the Interface Module will be included in the class. See Link.
Register	source-and-GP	Registers the class in the system registry. For more information, see the NaturalX documentation.
Unregister	source-and-GP	Unregisters the class from the system registry. For more information, see the NaturalX documentation.
Rename	new source-only source-and-GP	Changes either the class name or the class module name depending on the current view of the library workspace. For more information, see Renaming Class Members.
Delete	new source-only source-and-GP	Deletes the Natural module of the class (for source-only and source-and-GP) or only the internal structure of the class (new).
Restore	source-only source-and-GP	Removes all changes of the class which have not yet been saved. This command will close all list views of the class and collapse the class node in the library workspace.
Cut	source-only source-and-GP	Cuts the class module.
Copy	source-only source-and-GP	Copies the class module.
Paste	source-only source-and-GP	Pastes the class module.
Print	new source-only source-and-GP	Prints the source format of the current class structure.
Properties	new source-only source-and-GP	Opens the Properties dialog which shows class-specific information. For more information, see Node Properties.

Object Data Nodes

An object data node represents an object data area module or an inline object data definition. A class can have several object data nodes. If more than one object data node exists, you must take care to follow the correct object data sequence when you use these nodes in method implementations.

Types

Data Area

This type indicates that the object data is defined in a separate Natural module of type local data area or parameter data area. The name which is displayed in the node is the name of the Natural data area module.

Inline Data Definition

This type indicates that the object data is defined direct in the class source with a DEFINE DATA OBJECT statement. In this case, the object data has to be defined again in every method implementation which uses the object data. A node of this type is always named "Inline".

Commands

Command	available for	Description
Open	data area	Opens the data area module with the data area editor.
Edit	inline data definition	Opens a dialog which shows the contents of the inline data definition for editing.
List	data area	Lists the data area module.
Cat	data area	Catalogs the data area module.
Stow	data area	Stows the data area module.
Unlink	data area	Unlinks the data area module from the class, i.e. it is no longer used as Object Data Area for the class.
Rename	data area	Renames the Object Data Area link, i.e. uses another data area module as Object Data Area for the class. For more information, see Renaming Class Members.
Delete	inline data definition	Deletes the inline data definition from the class.
Print	data area	Prints the data area module.
Properties	data area inline data definition	Opens the Properties dialog which shows object data-specific information. For more information, see Node Properties.

GUID Local Data Nodes

An GUID Local Data node represents a local data area module or an inline local data definition which contains GUID definitions. A class can have several local data nodes.

Types

Data Area

This type indicates that the GUID local data is defined in a separate Natural module of type local data area or parameter data area. The name which is displayed in the node is the name of the Natural data area module.

Inline Data Definition

This type indicates that the GUID local data is defined direct in the class source with a DEFINE DATA LOCAL statement. A node of this type is always named "Inline".

Commands

Command	available for	Description
Open	data area	Opens the data area module with the data area editor.
Edit	inline data definition	Opens a dialog which shows the contents of the inline data definition for editing.
List	data area	Lists the data area module.
Cat	data area	Catalogs the data area module.
Stow	data area	Stows the data area module.
Unlink	data area	Unlinks the data area module from the class, i.e. the data area module is no longer used as GUID Local Data Area for the class.
Rename	data area	Renames the GUID Local Data Area link, i.e. uses another data area module as GUID Local Data Area for the class. For more information, see Renaming Class Members.
Delete	inline data definition	Deletes the inline data definition from the class.
Print	data area	Prints the data area module.
Properties	data area inline data definition	Opens the Properties dialog which shows local data-specific information. For more information, see Node Properties.

Interface Module Nodes

An Interface Module node represents an interface module. The interface module is a Natural module of type copycode which defines interfaces that can be included in several classes. For more information about interface modules and their usage, see Using Interfaces from several Classes.

Commands

Command	Description
Open	Opens the interface module list view. For more information, see List View.
List	Opens the program editor in read-only state with the source format of the current Interface Module structure.
Save	Saves the current Interface Module structure in the given Natural copycode module.
New Interface	Creates a new interface in the Interface Module.
Unlink	Unlinks the Interface Module from the class, i.e. the interfaces defined in the Interface Module are no longer available in the class.
Print	Prints the source format of the current Interface Module structure.
Properties	Opens the Properties dialog which shows Interface Module-specific information. For more information, see Node Properties.

Interface Nodes

An interface node represents an interface of an interface module or a class. For more information about internal and external interfaces, see Using Interfaces from several Classes.

Types

Internal

The parent of an internal interface is either an interface module or a class. If its parent is an interface module, this means that the interface is defined in the interface module which is used by the class. In this case, the interface will be displayed a second time as an external interface of the class (For more information, see Using Interfaces from several Classes). If the internal interface is a child of the class itself, this means that the interface is defined direct in the class.

External

An external interface can appear only as sub-node of a class, which uses an interface module which defines this interface. The commands which can be executed on an external interface node are only a subset of the commands available for an internal interface. Basically you can only change the implementation of such an interface. For more information, see Using Interfaces from several Classes.

Commands

Command	available for	Description
Open	internal external	Opens the interface list view. For more information, see List View.
New Method	internal	Creates a new method for the interface.
New Property	internal	Creates a new property for the interface.
Rename	internal	Renames the interface. For more information, see Renaming Class Members.
Delete	internal	Deletes the interface and all its dependent components.
Properties	internal external	Opens the Properties dialog which shows interface-specific information. For more information, see Node Properties.

Property Nodes

A property node represents a property of an internal or external interface.

Types

Internal

If a property appears as sub-node of an internal interface, it will be displayed as internal property. An internal property node always has a dedicated external property node.

External

If a property appears as sub-node of an external interface, it will be displayed as external property. The commands which can be executed on an external property are only a subset of the commands which are available for internal properties.

Commands

Command	available for	Description
Rename	internal	Renames the property. For more information, see Renaming Class Members.
Delete	internal	Deletes the property.
Properties	internal external	Opens the Properties dialog which shows property-specific information. For more information, see Node Properties.

Method Nodes

A method node represents a method of an internal or external interface.

Types

Internal

If a method appears as sub-node of an internal interface, it will be displayed as an internal method. An internal method node always has a dedicated external method node.

External

If a method appears as sub-node of an external interface, it will be displayed as external method. The commands which can be executed on an external method are only a subset of the commands which are available for internal methods.

Commands

Command	available for	Description
New PDA	internal	Creates a new method parameter data area for the method.
Link PDA	internal	Uses an existing parameter data area as method PDA. See Link.
Link implementation	internal external	Uses an existing subprogram as method implementation. See Link.
Rename	internal	Renames the method. For more information, see Renaming Class Members.
Delete	internal	Deletes the method and all its dependent components.
Properties	internal external	Opens the Properties dialog which shows method-specific information. For more information, see Node Properties.

Method Implementation Nodes

A method implementation node represents the Natural subprogram which is executed when the method is called.

Commands

Command	Description
Open	Opens the subprogram of the method implementation in the program editor.
List	Lists the subprogram of the method implementation in read-only mode in the program editor.
Cat	Catalogs the subprogram of the method implementation.
Stow	Stows the subprogram of the method implementation.
Rename	Renames the method implementation, i.e. uses another subprogram for the method implementation. For more information, see Renaming Class Members.
Print	Prints the subprogram of the method implementation.
Properties	Opens the Properties dialog which shows method implementation-specific information. For more information, see Node Properties.

Method Parameter Data Nodes

A method parameter data node represents a parameter data area module or an inline parameter data definition. A method can have several method parameter data nodes, which define the parameter used by the method implementation. If more than one method parameter data node exists, you must ensure that the correct parameter data sequence is used in method implementations.

Types

Data Area

This type indicates that the method parameter data is defined in a separate Natural module of type parameter data area. The name which is displayed in the node is the name of the Natural parameter data area module.

Inline Data Definition

This type indicates that the method parameter data is defined direct in the class source (or interface module source) with a DEFINE DATA PARAMETER statement. In this case, the parameter data must be defined again in every method subprogram. A node of this type is always named "Inline".

Commands

Command	**available for**	**Description**
Open	data area	Opens the data area module with the data area editor.
Edit	inline data definition	Opens a dialog which shows the contents of the inline data definition for editing..
List	data area	Shows the listing of the data area module.
Cat	data area	Catalogs the data area module.
Stow	data area	Stows the data area module.
Unlink	data area	Unlinks the data area module from the method, i.e. the data area module is no longer used as parameter data area for the method.
Rename	data area	Renames the method parameter data area link, i.e. uses another data area module as parameter data area for the method. For more information, see Renaming Class Members.
Delete	inline data definition	Deletes the inline data definition.
Print	data area	Prints the data area module.
Properties	data area inline data definition	Opens the Properties dialog which shows method parameter data-specific information. For more information, see Node Properties.

Node Properties

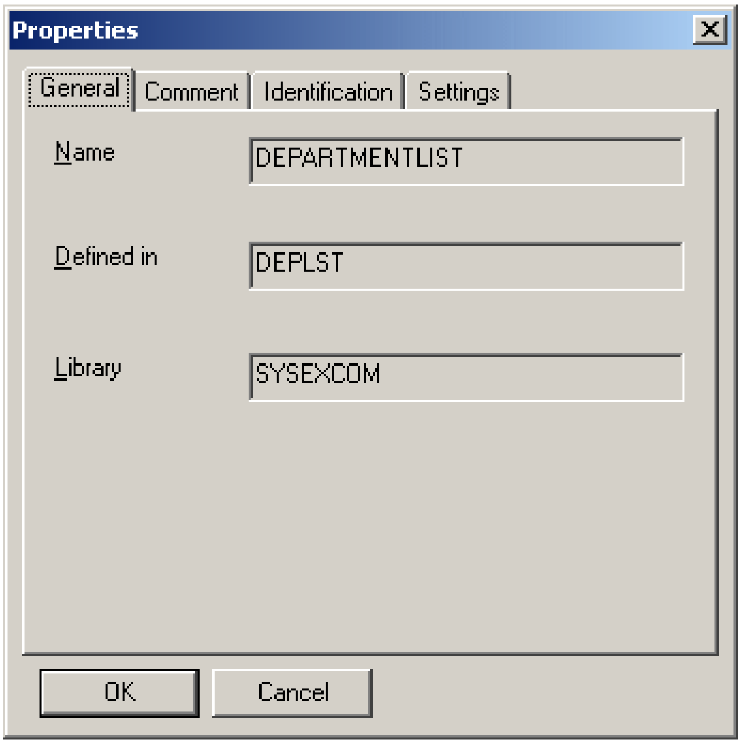
The Class Builder provides node-specific information on Natural classes and their elements if context-menu entry "Properties" is chosen. This context-menu entry is available if an object is selected in the library workspace or in a list view. The property sheet provides no information on group nodes.

The information itself is presented in a property sheet. The actual number of property pages shown depends on the type of the selected object.

● OK	Accept modifications.
● Cancel	Skip modifications.

For all class elements, property pages "General" and "Comment" are available. The other property pages depend on the selected node type.

General



Name: DEPARTMENTLIST

Defined in: DEPLST

Library: SYSEXCOM

This property page shows general information on the selected object. Its contents vary with the corresponding type of node and are described in the following sections.

Class

Name	Class Name
Defined in	Class Module
Library	Library

Object and Local Data Area

Name	Name of Object or Local Data Area
Used in	Class Name
Library	Library

Inline Data Definition

Name	"Inline Definition"
Defined in	Class Name

Interface Module

Name	Name of Interface Module
Used in	Class Name
Library	Library

Interface

Name	Name of Interface
Defined in	Class Name
Interface Module	If the interface is defined in an interface module this field shows the corresponding name.

Method

Name	Name of Method
Defined in	Name of the interface that offers this method.
Interface Module	If the method is defined in an interface module this field shows the corresponding name.

Implementation

Name	Name of Subprogram
Used in	Name of the method that is implemented by this subprogram.
Library	Library

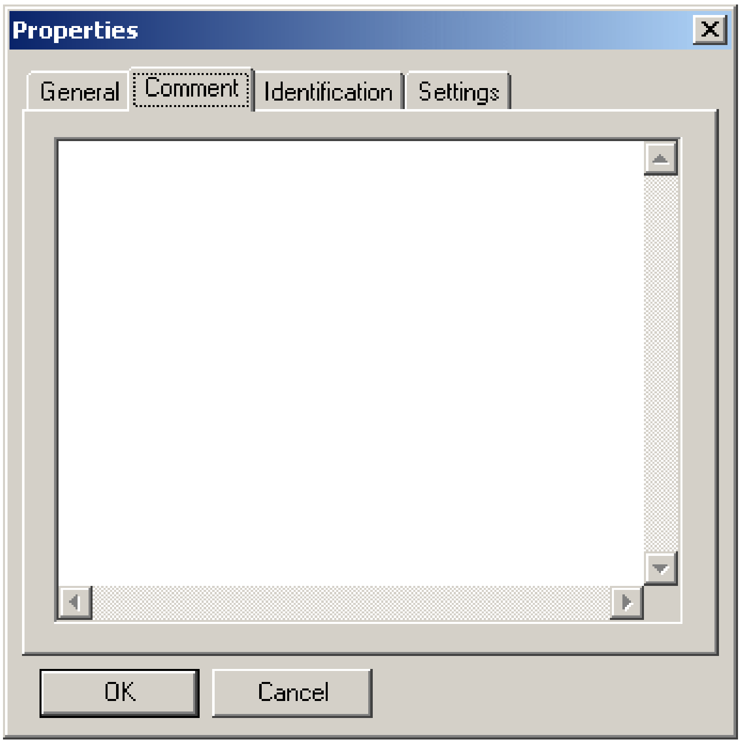
Parameter Data Area

Name	Name of Parameter Area
Used in	Name of Method
Library	Library

Property

Name	Name of Property
Defined in	Name of the interface that offers this property.
Interface Module	If the property is defined in an interface module this field shows the corresponding name.

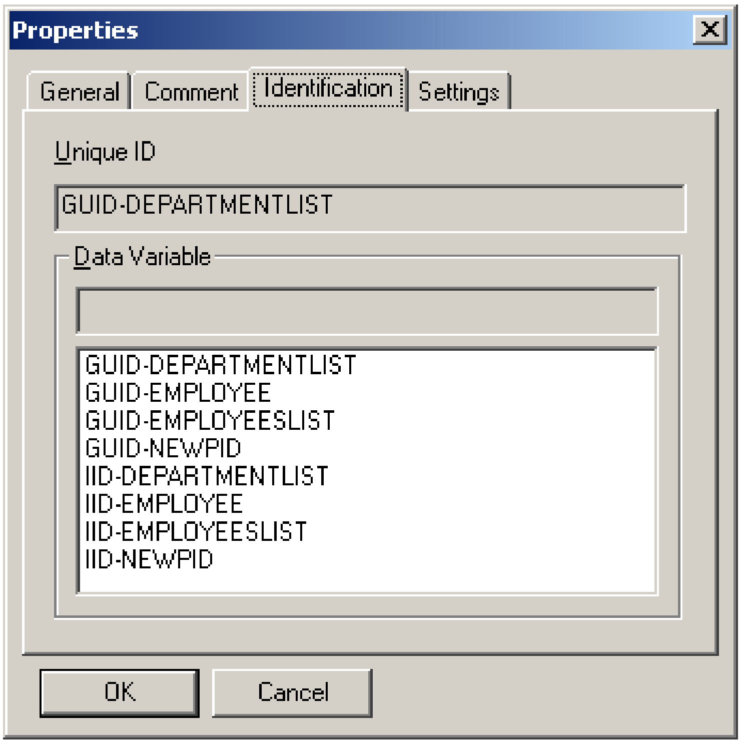
Comments



Each component has its own comment.
This property page shows the comment and allows adding new or modifying existing comments. They are entered and listed without any special syntactic notation.

The comment is changed if the property sheet is left by pressing "OK". Pressing "Cancel" leaves the comment unchanged.

Identification



Unique ID

GUID-DEPARTMENTLIST

Data Variable

GUID-DEPARTMENTLIST
GUID-EMPLOYEE
GUID-EMPLOYEESLIST
GUID-NEWPID
IID-DEPARTMENTLIST
IID-EMPLOYEE
IID-EMPLOYEESLIST
IID-NEWPID

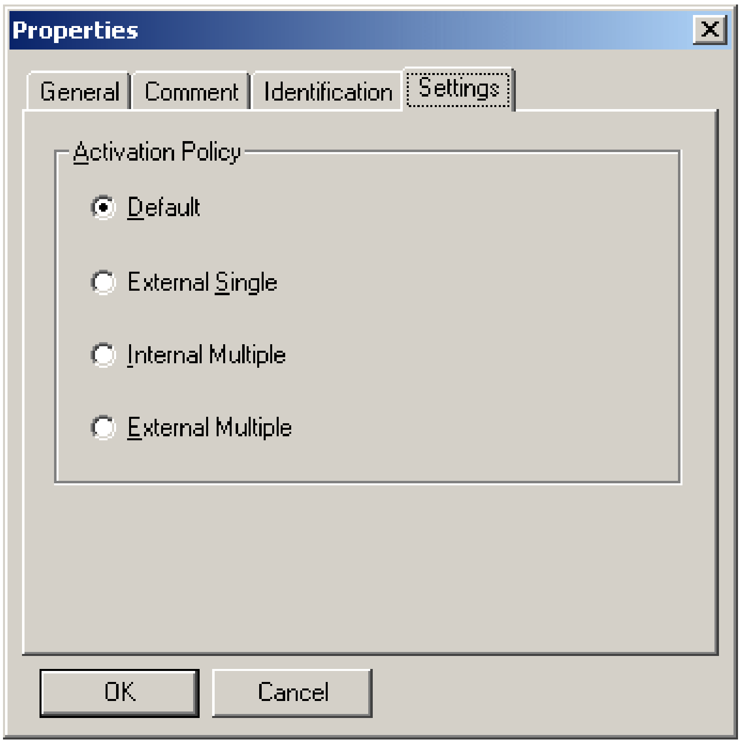
This property page is available for class and interface nodes. For interfaces, the list box below is only enabled if the interface is defined direct as part of the class. The list box is not visible if the interface is defined in an interface module.

The upper control "Unique ID" shows the current **G**lobal **U**nique **ID** of a class or an interface as read-only information.

This list box offers all data variables contained in local data areas that are linked to the class. These variables can be used as unique identifiers. Inline definitions of variables are not supported.

To exchange the current **G**lobal **U**nique **ID** that is displayed in the upper control with another value, select a variable from the list. The name control is then updated with the newly selected variable name. The **G**lobal **U**nique **ID** is exchanged if a variable has been selected and the property sheet is left by pressing "OK". Pressing "Cancel" leaves the identification unchanged. There is no check whether a selected variable represents a valid **G**lobal **U**nique **ID**.

Settings



Activation Policy

● Default

○ External Single

○ Internal Multiple

○ External Multiple

This property page is available for class nodes only. It allows setting the class"s activation policy within the Class Builder.

A class"s activation policy can be

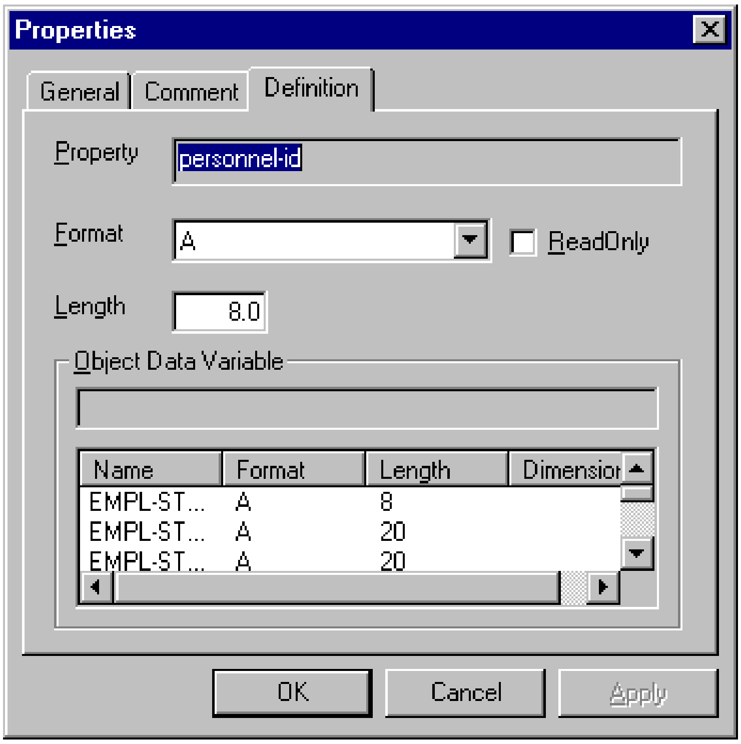
- External Single
- Internal Multiple
- External Multiple

Or it is set to default.

More information on the meaning of these values can be found in the NaturalX documentation.

To change the current activation policy select the required value.
The value is changed if the property sheet is left by pressing "OK". Pressing "Cancel" leaves the identification unchanged.

Definition



Property personnel-id

Format A

□ ReadOnly

Length 8.0

Object Data Variable

Name	Format	Length	Dimension
EMPL-ST...	A	8	
EMPL-ST...	A	20	
EMPL-ST...	A	20	

This property page is available for properties of interfaces only. It allows modifying the definition of an existing property.

The property"s name cannot be changed. The following changes are possible:

- An Object Data Variable can be assigned to the property.
  The available Object Data Variables are listed in the page"s list box together with their format definition and dimension.
  They are taken from the Object Data Areas that are linked to the current class. Inline definitions of variables are not supported.
- Existing assignments of Object Data Variables to properties can be changed. The corresponding control is then updated with the newly selected variable"s name.
- The property"s format definition can be added or changed if it is different from the Object Data Variable"s definition.
  Otherwise format and length definition are taken from the assigned Object Data Variable.
- It can be defined whether this property is used read only.

The definition of the property is changed if the property sheet is left by pressing "OK". Pressing "Cancel" leaves the definition unchanged.

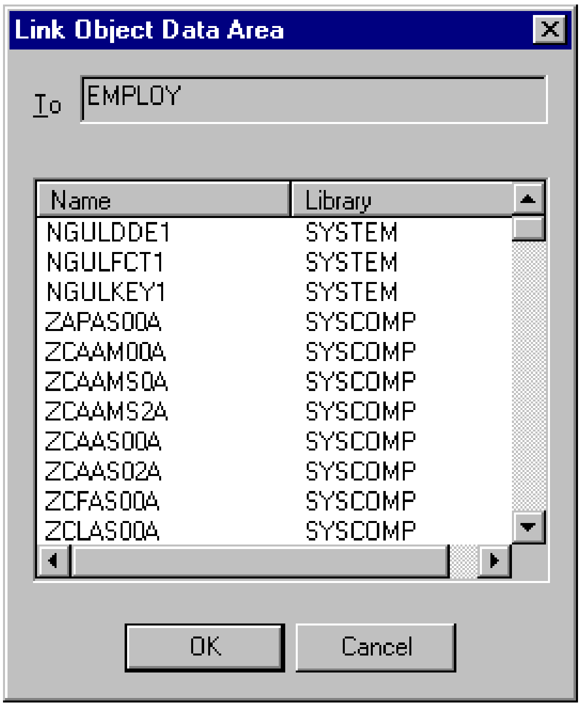
Adding Class Components

To make the development of a class more comfortable the Class Builder offers two ways to add components to a class.

Link

Existing Natural objects can be linked to a class component.
If context menu item "Link" is activated for an object node a dialog is opened. It lists all objects of the required type that can be found in the current library or its steplibs.

If an object has been selected and the dialog is left by pressing "OK", a reference to the selected object is added to the class structure. "Cancel" leaves the class structure unchanged.



To: EMPLOY

Name	Library
NGULDDE1	SYSTEM
NGULFCT1	SYSTEM
NGULKEY1	SYSTEM
ZAPAS00A	SYSCOMP
ZCAAM00A	SYSCOMP
ZCAAMS0A	SYSCOMP
ZCAAMS2A	SYSCOMP
ZCAAS00A	SYSCOMP
ZCAAS02A	SYSCOMP
ZCFAS00A	SYSCOMP
ZCLAS00A	SYSCOMP

Link to Class

A GUID Local Data Area, an Object Data Area or an Interface Module can be linked to a class. The dialog shows object name and library.

Link to Method

Each method requires a method implementation. The existing implementation can be exchanged by linking another subprogram to a selected method. Moreover, one or more Parameter Data Areas can be linked to a method. The dialog shows object name and library.

New

New class components are created with context menu item "New".

In the library workspace, class components are created using in-place editing. List views use dialogs to query the necessary data and create new objects. This applies to all nodes apart from class properties: They are always created using a dialog.

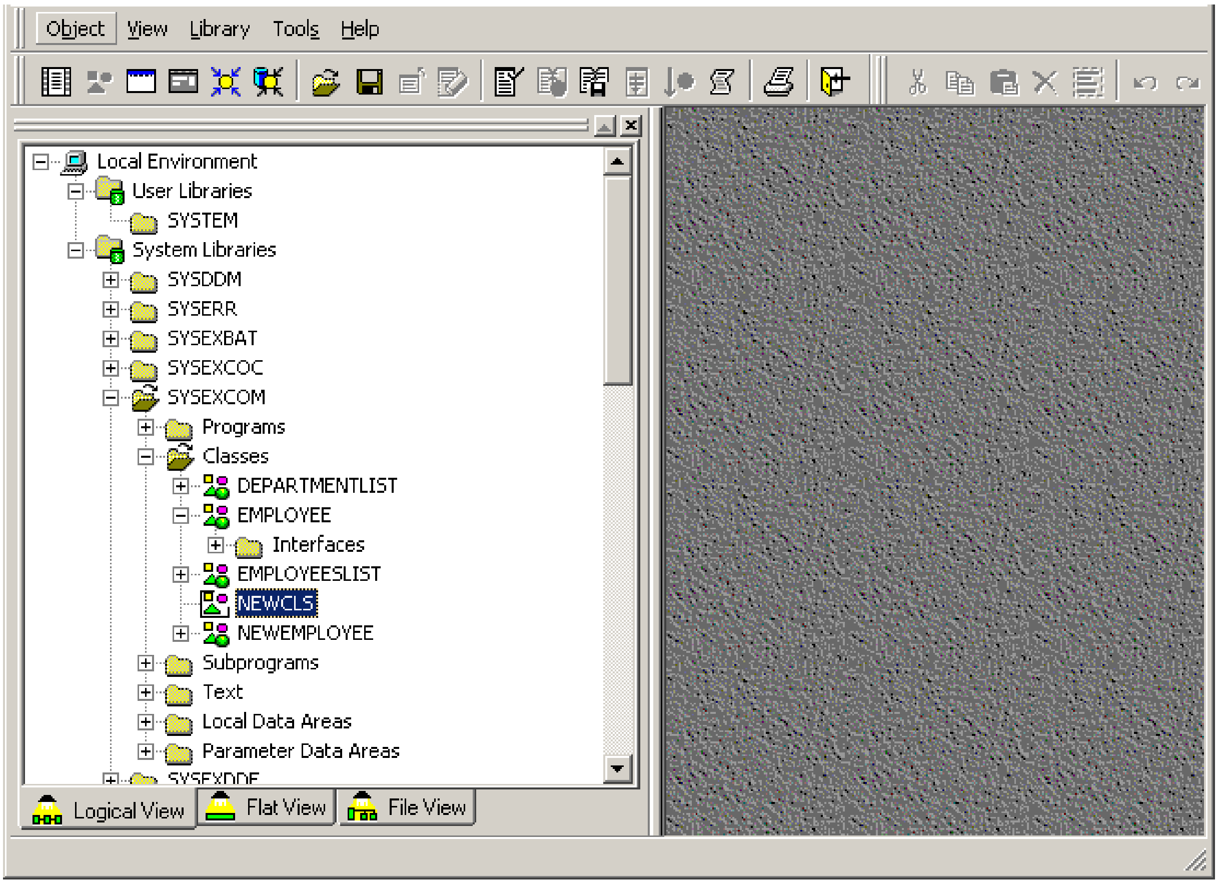
The following sections describe how the different class components are created.

New Class

A new class is first created as an internal class structure. At this time the class name is defined. The class module name, i.e. the name of the actual Natural object, is assigned when the class is saved the first time.

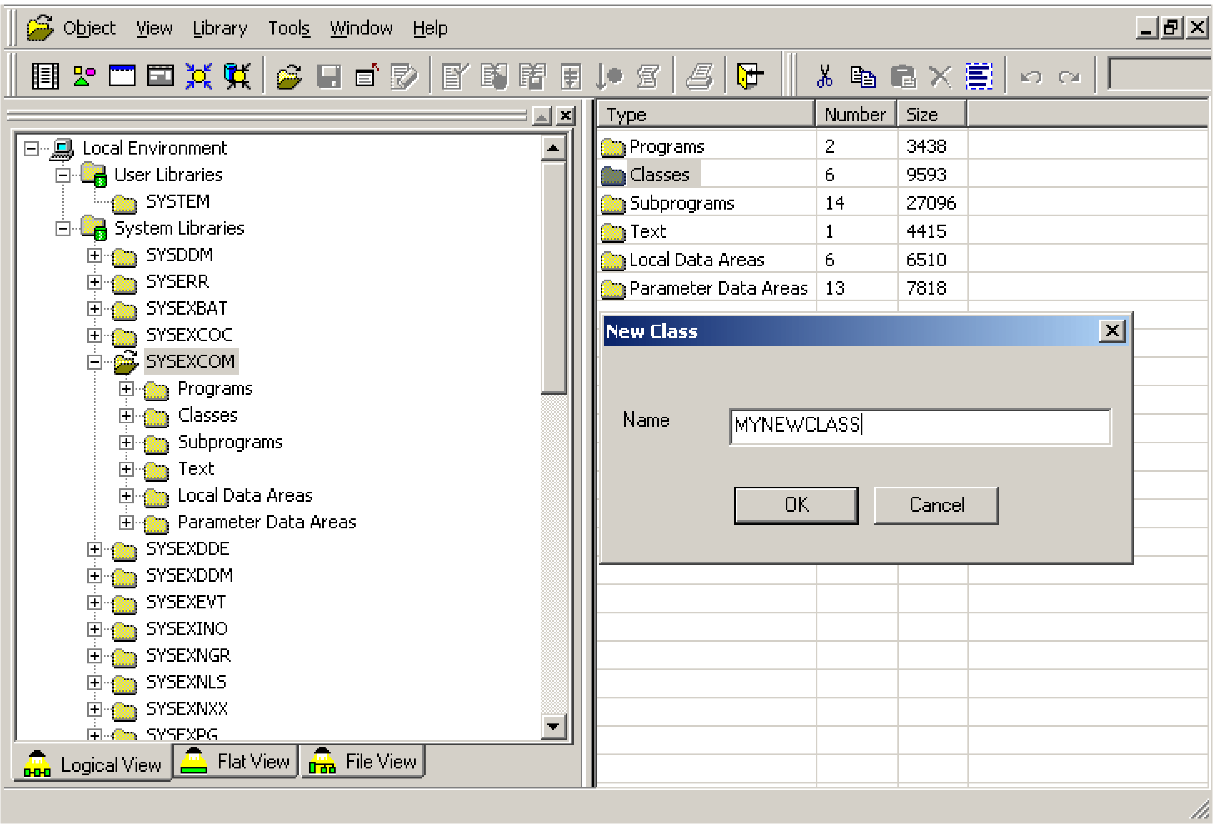
Library Workspace

A new class name, for example NEWCLS, is generated. The corresponding tree node is selected and made available for in-place editing. The name can be changed to any valid class name.



List View

A dialog is opened that asks for the name of the new class.



Type	Number	Size
Programs	2	3438
Classes	6	9593
Subprograms	14	27096
Text	1	4415
Local Data Areas	6	6510
Parameter Data Areas	13	7818

New Class

Name MYNEWCLASS

New Object Data Area

Creating a new object data area adds a reference to a new component to the class structure. The corresponding Natural object is not yet created. It is created if you confirm such when you open it.

Library Workspace

A new object data area, for example NEWODA, is generated. The corresponding node is selected and is made available for **in-place editing**. The name can be changed to any valid data area name.

List View

A dialog is opened that asks for the name of the new object data area.

New Interface Module

Creating a new interface module adds a reference to a new component to the class structure. The corresponding Natural object is not yet created. It is created if it contains interfaces at the time the class is saved.

Library Workspace

A new interface module, for example NEWEIF, is generated. The corresponding node is selected and is made available for in-place editing. The name can be changed to any valid copycode name.

List View

A dialog is opened that asks for the name of the interface module.

New Interface

Library Workspace

A new interface, for example NEWIIF, is generated. The corresponding node is selected and is made available for in-place editing. The name can be changed to any valid interface name.

List View

A dialog is opened that asks for the name of the interface.

New Method

Library Workspace

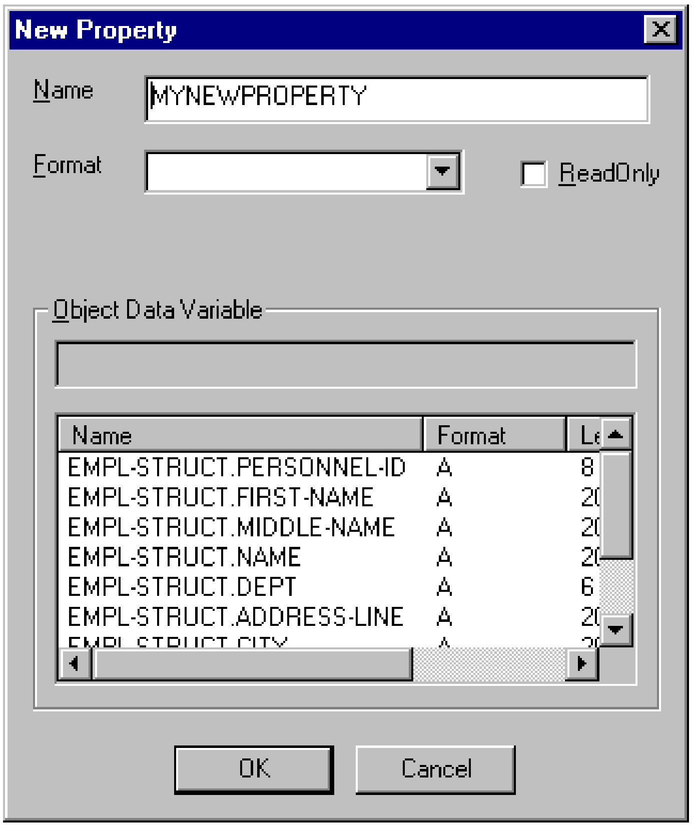
A new method, for example NEWMET, is generated. The corresponding node is selected and is made available for in-place editing. The name can be changed to any valid method name. The new method name is also taken as the name of the method implementation. Both are added to the class structure. If the method name is longer than a valid Natural subprogram name, only the first characters are used to guarantee a valid implementation name.

List View

A dialog is opened that asks for the name of the method. The new method name is also taken as the name of the method implementation. Both are added to the class structure. If the method name is longer than a valid Natural subprogram name the first characters are used to guarantee a valid implementation name.

New Property

New properties are always created using a dialog.



Name: MYNEWPROPERTY

Format:

□ ReadOnly

Object Data Variable

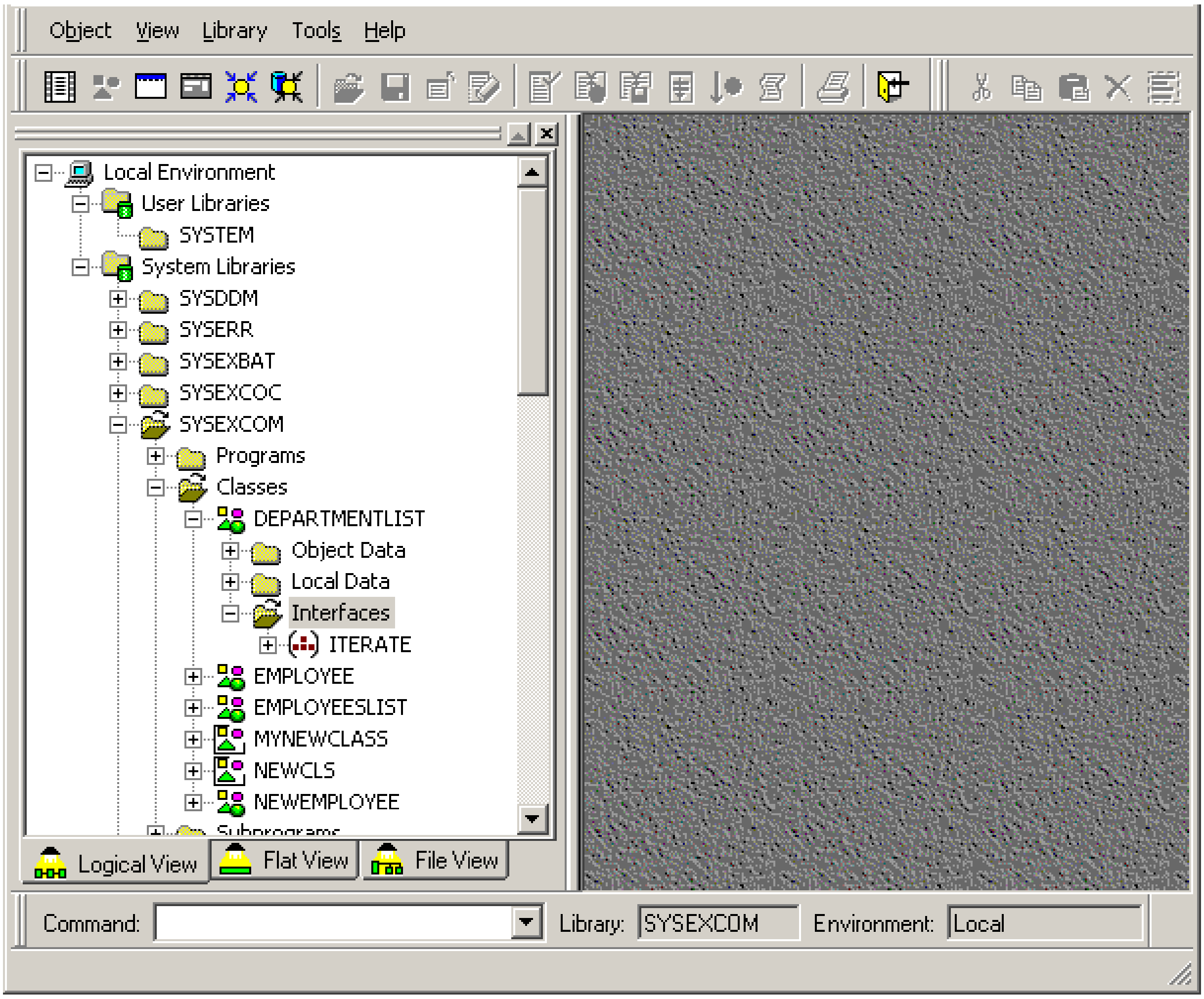
Name	Format	Le
EMPL-STRUCT.PERSONNEL-ID	A	8
EMPL-STRUCT.FIRST-NAME	A	20
EMPL-STRUCT.MIDDLE-NAME	A	20
EMPL-STRUCT.NAME	A	20
EMPL-STRUCT.DEPT	A	6
EMPL-STRUCT.ADDRESS-LINE	A	20
EMPL-STRUCT.CITY	A	20

This dialog retrieves the following information:

Property name:	A valid property name. This is either a new name or the name of the selected ODA variable. For fully qualified ODA variable names the dot is replaced by an underscore.
ODA variable:	The list box lists all variables that are defined in the linked ODAs. If property name, format and length are not changed, these values are taken from the selected ODA variable.
Format:	Format and length can be changed if they must be different from the ODA variable"s definitions.
Read-Only:	The property can be marked as read-only.

Renaming Class Components

Like any other Natural object that can be modified in the Natural Studio, the components of a class are renamed by editing their identifier in place. This is done using the mouse or by pressing F2 or by choosing context menu entry "Rename" which is enabled for every class component.



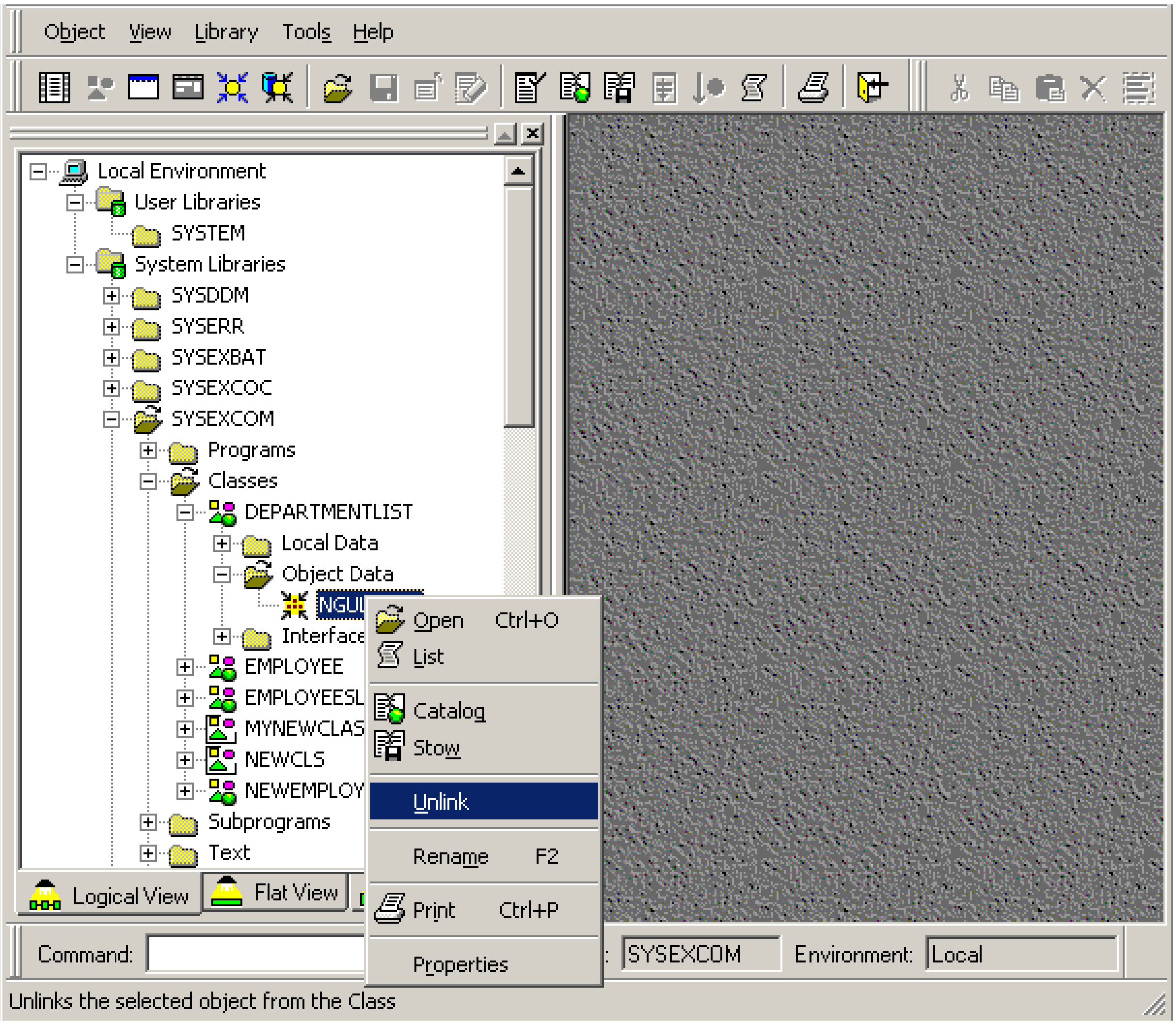
During the edit process the new name is checked for syntactical correctness. If it is not a valid Natural name the edit mode cannot be left. Pressing ESCAPE cancels the edit mode and resets the old identifier.

If class components refer to Natural objects such as Object Data Areas, Parameter Data Areas or Interface Modules, only the references within the class are changed. The corresponding Natural objects are not renamed. They have to be changed explicitly if required.

Removing Class Components

Unlink

Context menu entry "Unlink" is available for class components that refer to Natural objects like Data Areas or Interface Modules. If these modules have been linked to a class previously they can be removed using "Unlink".



Unlinks the selected object from the Class

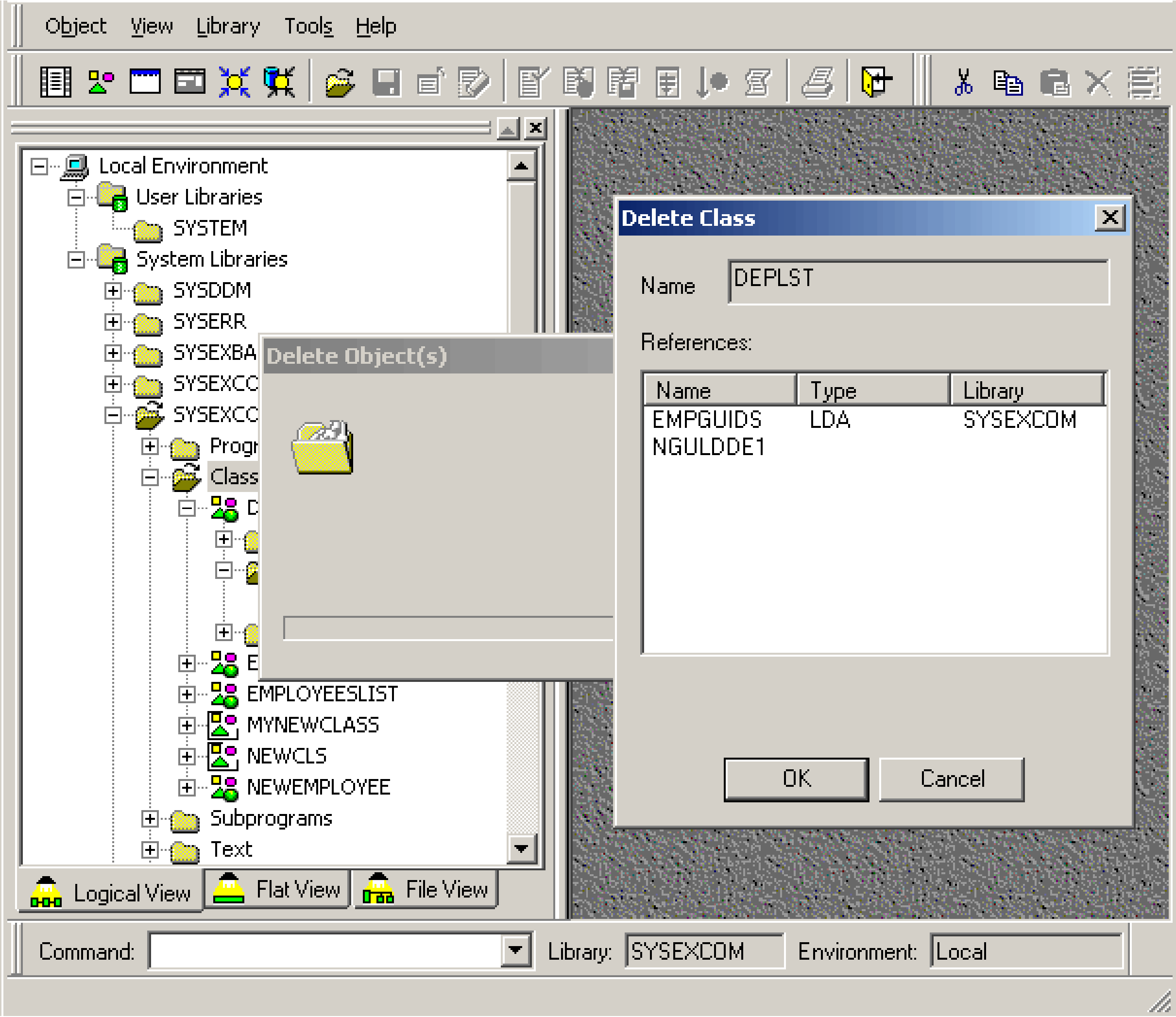
This action only removes the reference to selected components from the class. It does not delete an existing Natural object.

Delete

Context menu entry "Delete" is available for classes and those of their components that do not refer to Natural objects.

If this context menu item is selected, a dialog is displayed.
It displays the name of the selected component in a read-only field together with a list of references that shows the dependent Natural objects. These objects are identified by name, library and Natural object type if required. The list serves for information purposes only. The dependent Natural sources are not affected.

If the selected component is the class itself, the internal structure is deleted and the corresponding Natural source and cataloged modules are removed from the library.



Delete Class

Name: DEPLST

References:

Name	Type	Library
EMPGUIDS	LDA	SYSEXCOM
NGULDDE1		

OK	Closes the dialog and deletes the selected component. If the selected component is the class itself, the internal structure is deleted and the Natural source and GP objects are removed from the library. The referenced Natural objects are not deleted.
CANCEL	Closes the dialog without deleting anything.

Editing Class Components

Classes

At the time a new class is created, the corresponding new class module is not yet created. This occurs only if "Save", "Save As" or "Stow" is called for the class.

Save

"Save" called for an existing class writes the class source to the class module.
If "Save" is called for a new class that does not yet have a corresponding class module then "Save" is treated like "Save As...". If such a class module does not yet exist in the current library, the class module is created and the source is written to this object.

Save As

If "Save As..." is called, a dialog is opened that prompts for the class module. The input length is restricted to guarantee a valid Natural class module name and the input is checked for validity. If such a class module does already exist or if the name is invalid, an error message is issued.

Cat

If the command "Cat" is called, the class source is cataloged and a corresponding class GP is generated. This does not apply to new classes.

Stow

As for other Natural objects "Stow" internally saves and catalogs a class. If a new class is to be stowed, you are prompted for the class module as described for "Save As".

Natural Objects

Natural objects that can act as class components can also be modified in the context of the class structure.
References to Object Data Areas, Parameter Data Areas and Interface Modules can be created by "New".
Existing objects can be edited, saved and stowed.
Local Data Areas and method implementations cannot be created in the class"s context. Here only existing objects can be linked to the class. But they can be edited, saved and stowed.

Other Class Components

Other class components such as interfaces, methods and properties cannot be saved, cataloged or stowed independently. They can only be modified in the context of a class.

Using Interfaces from Several Classes

For some applications, it is useful to implement the same interface in several classes. For this purpose, it is possible to define the interface in a Natural copycode module and include this copycode module in the class which wants to implement the interface. The implementation-specific settings, like method implementations, can be defined in the copycode as a default setting, and they can be overwritten in the class, to use class specific implementations.
Natural copycode modules which define interfaces are called Interface Modules in the Class Builder environment. Interface Modules are fully integrated in the Class Builder, so that interfaces defined in an Interface Module can be handled in the same way as interfaces of a class. However, an Interface Module can only be changed with the Class Builder when it is included from a class.

Interfaces which are defined in an Interface Module are always visible in two places of a class: they are shown as an internal interface under the Interface Module node and they are shown as an external interface under the class node. The commands available for an external interface can be used to change the implementation of the interface.

You can save a changed Interface Module without saving the whole class. If an Interface Module is changed and the class which is the parent of the Interface Module node is saved, the Class Builder asks the user if he wants to save the Interface Module as well.

The locking principles for Interface Modules are described in Locking Concept.

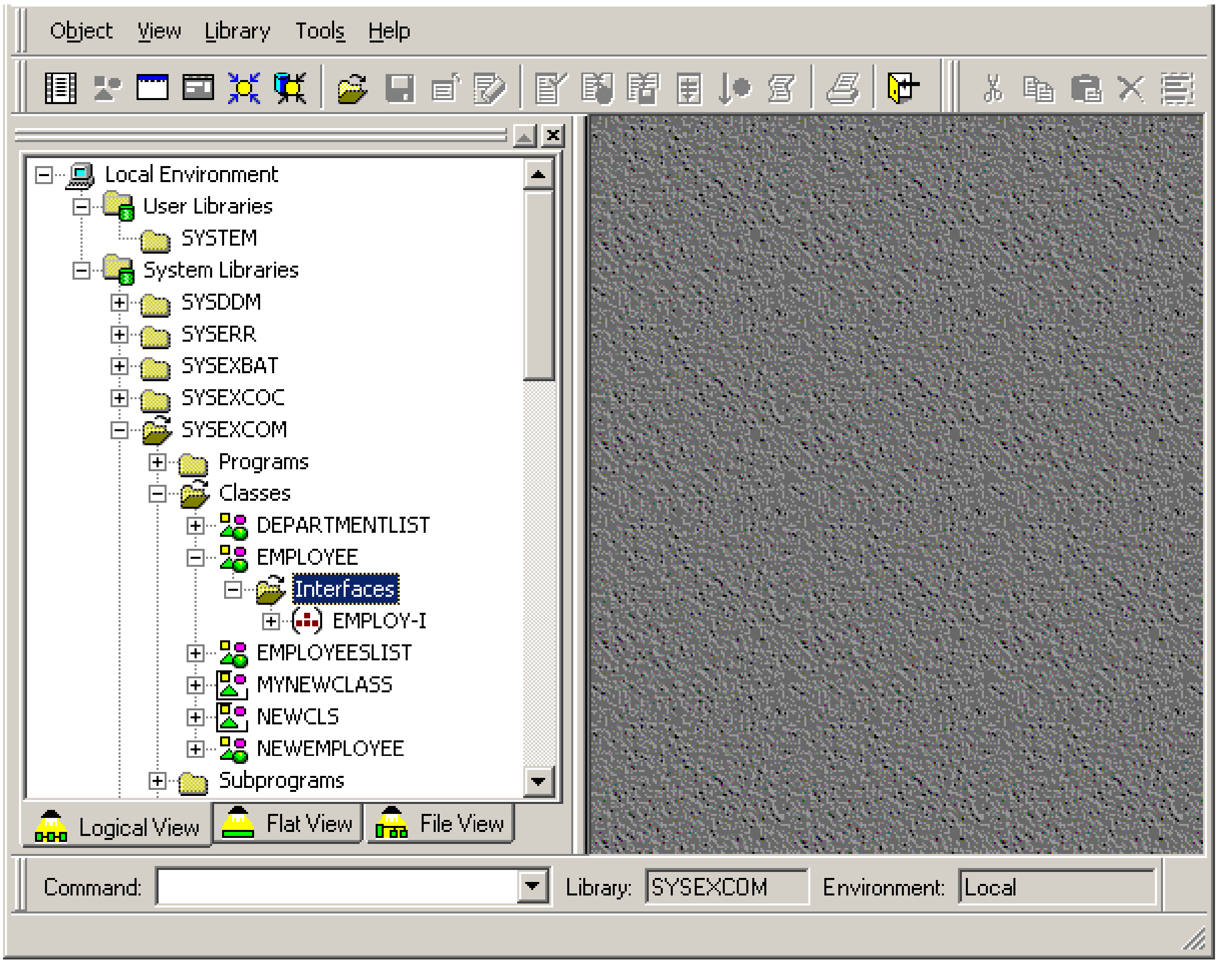
Note: If you change an Interface Module, you should always be aware that this Interface Module can also be used by other classes. After saving the changes other classes can possibly no longer be stowed without errors. The Class Builder cannot check if your Interface Module is used by other classes!

Creating a new Interface Module

The class command "New Interface Module" (see Class Builder Nodes) creates a new Interface Module.
An Interface Module node is added in the tree and list views and you can then create new interfaces for the Interface Module, methods and properties for the interfaces and so on. If a new component is created for the Interface Module, the corresponding external node will be added for the class. For example, if a new interface INT1 as added to the Interface Module, an external interface node named INT1 will be created as sub-node of the class. The new Interface Module is saved just as an existing Interface Module. As soon as the Interface Module exists as Natural module, it can be linked from other classes.

Linking an existing Interface Module

The class command "Link Interface Module" (see Class Builder Nodes) uses an existing Interface Module for the class. A dialog is shown which lists all Natural copycode modules of the current step libraries (**Note:** the dialog will list all copycode modules and not only the Interface Modules). If you select a copycode module from this list which defines class interfaces, these interfaces are added to the current class interfaces. An error will be generated if you select a copycode module which does not define interfaces or if the selected copycode module contains an interface which is already defined in the class. In this case, the Interface Module is not linked to the class.



If the Interface Module was linked successfully to the class, a node for it will be added to the class tree. Opening the Interface Module node will show the interfaces of the Interface Module. Furthermore all interfaces of the Interface Module are added as external interfaces nodes to the class itself.

Unlinking an Interface Module

If the "Unlink" command (see Interface Module Nodes) is executed for an Interface Module, the interfaces of this Interface Module are no longer used by the class.
This has the effect that the Interface Module node itself and all external interface nodes from this Interface Module are removed from the class.

Note:
If you unlink an Interface Module from a class, all class-specific settings contained in the class source module, such as method implementations for the interfaces of this Interface Module, will be deleted as well.

Interface Nodes

If an Interface Module is used by a class, every interface defined in the Interface Module is represented by two nodes: an internal interface node which is a sub-node of the Interface Module and an external interface node which is a sub-node of the class. These two interface node types can be distinguished by their icon (see Interface Nodes). The same is of course valid for the property and method nodes: if they are children of an internal interface, they are represented by an internal node and if they are children of an external interface, they are represented by an external node (see Property Nodes and Method Nodes).
Furthermore the commands which can be executed on external interfaces, properties and methods are only a subset of the commands available on internal interfaces, properties and methods. For example, the name of an interface can only be changed for an internal interface. External interfaces allow only the redefinition of the implementation of the interface, i.e. changing the method implementation and the ODA variable which is assigned to a property.

Locking Concept

Natural must ensure that a Natural module cannot be changed at the same time from different places. Therefore, related to the Class Builder, this means that a Natural user must be prevented from changing a Natural module with the program editor which has already been changed with the Class Builder and vice versa.
The Class Builder can be used to change Natural classes and Interface Modules which are special copycode modules (see Using Interfaces from several Classes).
Because of the different requirements, the locking concept for classes differs from the Interface Module locking concept. In the following sections both concepts are described.

Locking of Classes

The locking of classes is done very flexibly. The Class Builder does not lock a class until it is changed. This means that a class which is opened with the Class Builder can be opened in the program editor as well.
If a class is opened in the program editor, the class nodes can be viewed in the Class Builder, but it is not possible to apply any changes. Before changing the class, the program editor session has to be closed first.
If a class is visible in the Class Builder and the user changed the class in the program editor, the changes will also be shown in the Class Builder when the class is saved. If a class has been changed with the Class Builder it is no longer possible to open this class with the program editor.

Locking of Interface Modules

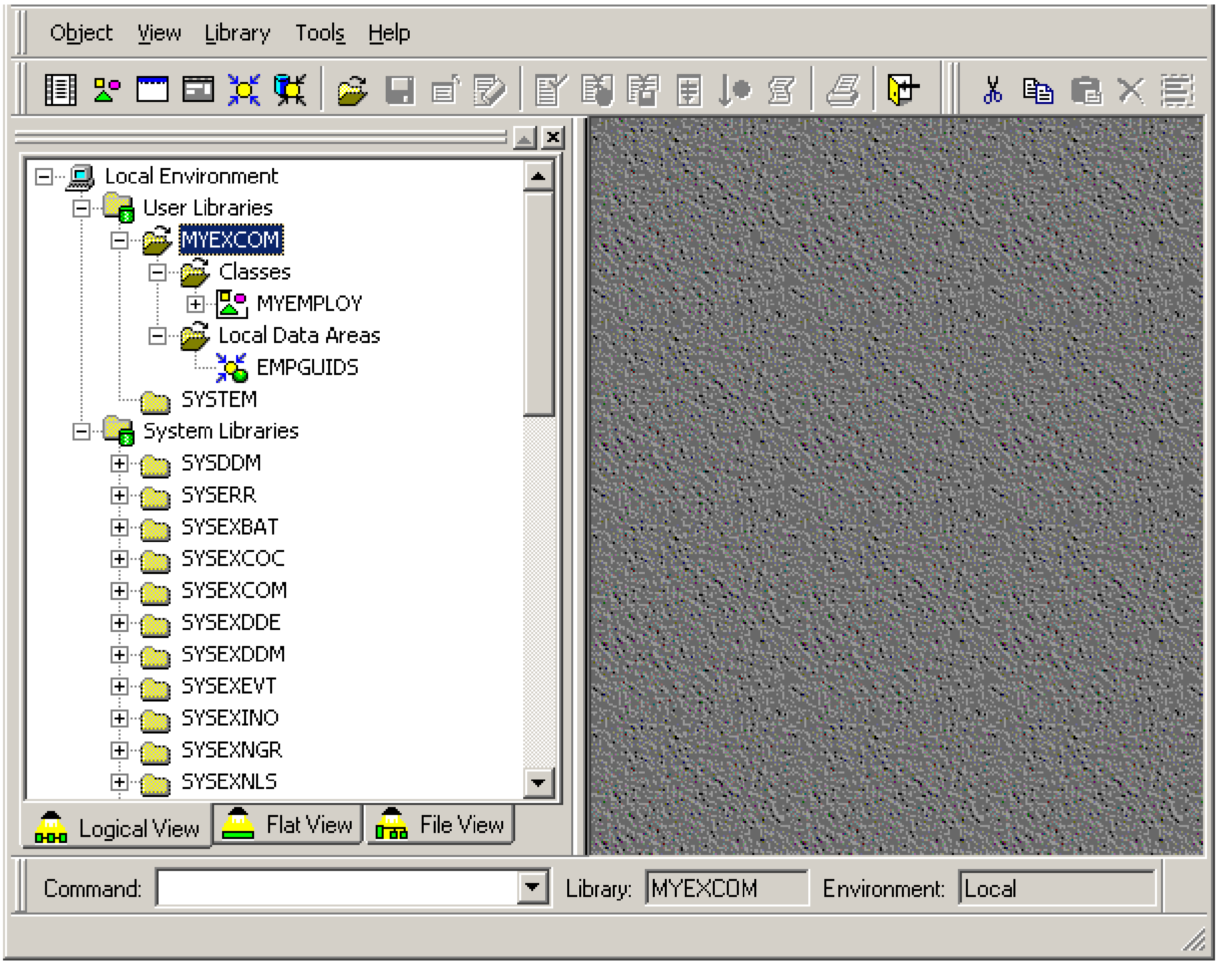
The locking of Interface Modules is a bit more restrictive than the locking of classes. A two stage locking exists for the Interface Modules. For the first time the Class Builder must ensure that the Interface Module cannot be changed with the Class Builder and the program editor at the same time: if a class which uses an Interface Module is opened in the Class Builder, the Interface Module is locked. This means on the one hand, that an Interface Module can no longer be opened with the program editor, when a class which uses it is opened in the Class Builder. On the other hand, a class cannot be opened with the Class Builder when it uses an Interface Module which is already open in the program editor.
Moreover, an Interface Module can be opened several times in the Class Builder if it is included from several classes. The Class Builder must ensure that an Interface Module is opened only once, when the user wants to change it, because the other Interface Module instances are then no longer up-to-date: it will try to close all other instances, to make sure that only the current instance of the Interface Module remains visible. The Class Builder will display a confirmation dialog for this purpose which allows the user to stop the process.
If one of the classes was already changed, the user will be asked, if the changes are to be saved . After saving a changed Interface Module, it is again possible to open other classes which use the Interface Module.

Tutorial

This chapter provides a short introduction on the usage of the Class Builder.
The example shows how class EMPLOYEE in library SYSEXCOM can be built using the Class Builder.

New class

Activate the logical view in the library workspace and create a new library MYEXCOM that contains the local data areas EMPGUIDS and EMPLOY-O. These are just copies of the objects in SYSEXCOM.
EMPGUIDS contains GUID definitions and EMPLOY-O contains object data definitions. To create a new class MYEMPLOYEE select the library node and then select context menu item "New > Class". A new tree node labeled "NEWCLS" is presented for in-place editing. Just change its name to "MYEMPLOYEE".



Library: MYEXCOM
Environment: Local

Linking Object Data

The object data for MYEMPLOYEE have to be defined in an object data area. This object data area can either be created by selecting context menu item "New" of node "MYEMPLOYEE" or by linking an existing object data area via context menu item "Link > Object Data Area...".
A dialog pops up and shows a list of all local and parameter data areas in MYEXCOM and its steplibs. These objects can be used as object data areas. Select EMPLOY-O.

Creating an Interface

To create the first interface select context menu item "New > Interface" of node "MYEMPLOYEE". A new tree node labeled "NEWIIF" is presented for in-place editing. Just change its name to "EMPLOY-I". Further interfaces can be created accordingly or by selecting "New" in the context menu for "Interfaces" (group node).

Creating Methods

To create the first method select context menu item "New > Method" of interface node "EMPLOY-I". A new tree node labeled "NEWMET" is presented for in-place editing. Rename this node to "INIT". A method implementation node with the same name is created automatically.
To use subprogram ELOAD-N (copied from SYSEXCOM) to implement this method, select the method"s context-menu item "Link > Implementation..." and change the method implementation.
Parameter Data Area ELOAD-A (copied from SYSEXCOM) can be linked using "Link > Parameter Data Area..." and then selecting the appropriate module. Further methods can be created accordingly or by selecting "New" in the "Methods" (group node) context menu.

Creating Properties

To create the first property, select context-menu item "New > Property..." of interface node "EMPLOY-I". The dialog lists all object data variables that are defined in linked object data areas and can be assigned to a property. They are shown together with their format and length definition and dimension. If one of these variables is selected without entering any information in the other control, this variable name is taken as property name and format and length definition are generated accordingly.
But the Class Builder allows assigning the property another name and format and length can be adapted as long as the new format is data-transfer compatible (see the NaturalX documentation). The new property can be marked as read only.

Using an Interface Module

So far class MYEMPLOYEE only defines interfaces internally. But there might be interfaces defined in modules that were adequate to incorporate.
For this purpose an interface module can be linked using the Class"s context-menu item "Link > Interface Module...". The interfaces that are defined in this module are then inserted under the corresponding interface module in group "Interface Modules" and at the same time under the group node "Interfaces". To implement their methods, select the corresponding node that can be found under "Interfaces".

Linking a GUID Local Data Area

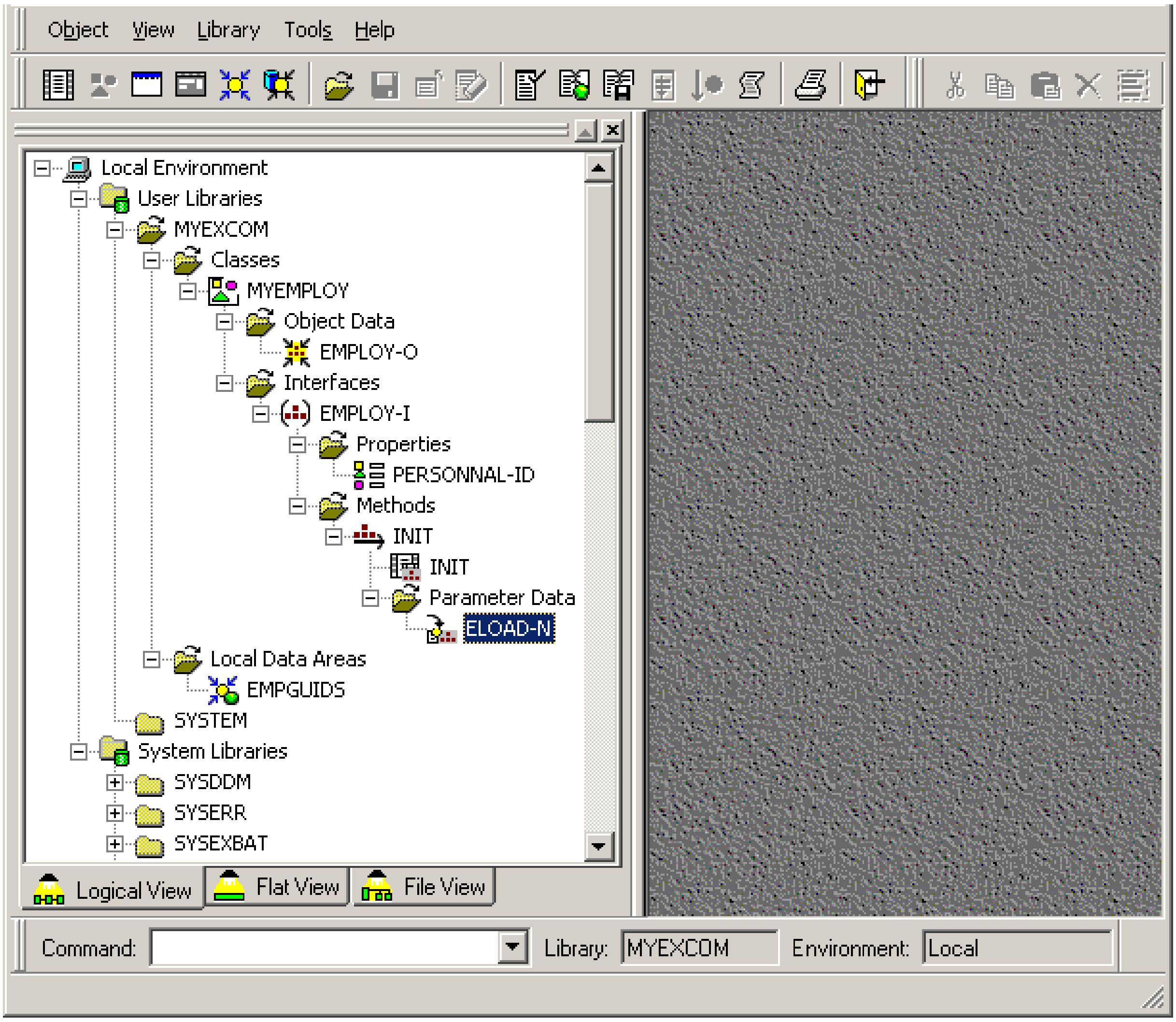
The Class Builder generates Global Unique IDs for classes and interfaces automatically. But if variables are to be used instead of the generated identifiers, a local data area with the corresponding definition can be linked to MYEMPLOYEE.
The existing Global Unique ID of MYEMPLOYEE can then be changed. Select context menu item "Properties" and activate page "Identifiers". This page is available for classes and interfaces.

The generated GUID is displayed in the upper control. Local variables that are defined in EMPGUIDS are listed in the lower box. Select EMPGUID and leave the property sheet with OK.

Activation Policy

The Class Builder allows setting a class"s activation policy explicitly. The current activation policy of MYEMPLOYEE can be viewed under "Settings" if context menu item "Properties" is selected. This option is available for classes only. Select "External Multiple" and leave the property sheet with OK.

Save and Stow Class



Command:

Library: MYEXCOM

Environment: Local

Up to now the new class MYEMPLOYEE has only existed as an internal class structure. To save all changes the class can be saved and stowed in the class module. This change of state is indicated by the changed icon.

Register

And finally register MYEMPLOYEE by selecting context menu item "Register" on the class node.

Glossary

External Interface

An external interface is an interface which is defined in an interface module, that is included by the class.

Interface **M**odule

An Interface Module is a Natural copycode module which defines interfaces. The Interface Module can be used in a class to define the contained interfaces. The class can overwrite the method and property implementations, but all other settings of the interface are used as defined in the Interface Module.

Internal Interface

An internal interface is an interface which is defined direct in the class, or an interface of an Interface Module, which is defined in the Interface Module.

Method Implementation

A method implementation is a Natural subprogram which is assigned to the method and executed when this method is called for a class object.

Property Implementation

A property implementation is the object data variable that is assigned to a property.

Program Editor

You use the program editor to write and maintain Natural programs, subprograms, subroutines, classes, copycode, helproutines and text elements. You can open multiple editing sessions, making it possible to copy or move content from one object to another. The title bar of each editing session displays the name and type of the object. If the object is new and has not yet been saved, it is automatically given the title "Untitled".

The following topics are covered below

- Modifying Program Contents
- Finding Source Contents
- Editing/Listing Referenced Natural Objects
- Splitting the Editor Window
- Expanding/Collapsing Object Listings
- Recording/Replaying Keystrokes
- Using Context-Sensitive Help
- Setting Editor Options

See also:

- Program Editor Accelerators
- Renumbering of Source-Code Line Number References

Modifying Program Contents

- Selecting Text
- Copying Text
- Cutting Text
- Pasting Text
- Deleting Text
- Undoing/Redoing Text Changes
- Renumbering a Program

Selecting Text

To select a single word using the mouse

- Double-click the word.

To select a range of text using the mouse

1. Point to the first character to be selected.
2. Drag the cursor to the last character you want to select.
3. Release the mouse button.

To select text using the keyboard

1. With the arrow keys, move the selection cursor to the first character to be selected.
2. Press and hold down **SHIFT** and use the arrow keys to select the text.

To select the entire editor contents

- From the **Edit** menu, choose **Select all**.

To cancel a mouse selection

- Click anywhere in the document.

To cancel a keyboard selection

- Press an arrow key.

Copying Text

Text can be copied within the same object or between two different objects.

To copy text

1. Select the text using the instructions provided in Selecting Text.
2. From the **Edit** menu, choose **Copy.**
   Or click the **Copy** toolbar button.
   Or press **CTRL+C**.
   The text is copied to the clipboard and can now be pasted within the same object or another object. For instructions on pasting text, see Pasting Text.

Cutting Text

The cut function can be used to delete text from an object or to move text within/between objects. When text is cut, it is taken from the object and placed on the clipboard. It remains there until the next cut or copy operation is performed, at which time it is irretrievably discarded from the clipboard to make way for the next cut/copied text.

It is possible to revoke a cutting operation after it has been performed. See Undoing/Redoing Text Changes.

To cut text

1. Select the text using the instructions provided in Selecting Text.

From the **Edit** menu, choose **Cut**.
Or click the **Cut** toolbar button.
Or press **CTRL+X**.
The text is cut to the clipboard and can now be pasted within the same object or another object. For instructions on pasting text, see Pasting Text.

Pasting Text

The paste function is used to place text at a specific position within an editor after it has been copied or cut to the clipboard from another position within the same object or another object. A text which has been copied or cut to the clipboard can be pasted repeatedly without recopying it.

It is possible to revoke a paste operation after it has been performed. See Undoing/Redoing Text Changes.

To paste text

1. Copy or cut a portion of text as described in Copying Text and Cutting Text.
2. Select with the I-beam pointer the position in the text where the text is to be inserted.
3. From the **Edit** menu, choose **Paste**.
   Or click the **Paste** toolbar button.
   Or press **CTRL+V**.
   The text is pasted to the object.
4. To paste the same text again, repeat Steps 2 and 3.

Deleting Text

When text is deleted, it is cut from the object but is **not** placed on the clipboard. The only way to recover deleted text is by undoing the deletion. See Undoing/Redoing Text Changes. Note that when text is deleted in the program editor, no warning is provided. This is intentional.

To delete text

1. Select the text using the instructions provided in Selecting Text.
2. From the **Edit** menu, choose **Delete**.
   Or click the **Delete** toolbar button.
   Or press **DEL**.

The text is deleted.

Undoing/Redoing Text Changes

The text operations you perform in the program editor can be revoked by applying the undo function or reinstated (after being undone) by applying the redo function. Typical operations which can be undone/redone are character input, character deletion, text deletion, text pasting, search/replace, or any action which modifies editor contents. The number of operations which can be undone/redone is determined by the value specified in the preferences for the program editor and is limited by the memory allocated (for more information, see Setting Editor Options).

The commands SAVE, CLOSE and CLEAR cause the undo/redo buffer to be cleared of its contents. A successful STOW also clears the buffer of its contents.

Undo / redo operations restore line numbering to its status before the operation.

To undo a text operation

- From the **Edit** menu, choose **Undo**.
  Or press **CTRL+Z**.
  Or click the **Undo** toolbar button.
  The text is restored to its condition before the previous text operation or redo operation.

To redo a text operation which has been undone using an undo operation

- From the **Edit** menu, choose **Redo**.
  Or press **CTRL+Y**.
  Or click the **Redo** toolbar button.
  The text is restored to its condition before the previous undo operation.

Renumbering a Program

As you add lines to a program, Natural numbers the added lines in increments of one. You never have to worry about two lines being assigned the same number. Natural recognizes such conditions and renumbers the program accordingly.

You can renumber a program at any time during an editing session. Every line of the object is renumbered, beginning with 0010 at the first line and increasing by increments of ten for each line.

To renumber a program

- From the **Edit** menu, choose **Renumber**.
  Or, in the command line, type "Renumber" and press **ENTER**.
  The program is renumbered.

Finding Source Contents

Searching for Source Text

In large objects, it is often difficult to locate a section of source code. Using the search function, you can flexibly search for any character string in source listings. If it should be necessary to replace a frequently occurring text string with another, you can use the combined search and replace function.

Searching for Source Text

To search for a text string in the active program window

1. From the **Edit** menu, choose **Find**.
   Or click the **Find** toolbar button.
   Or press **CTRL+F**.
   The "Find" dialog box appears.
2. In the "Find" text box, enter the string to be searched for.
3. If you want the search to be case sensitive, select the "Case Sensitive" check box.
   The search will then only find the string exactly as it appears in the "Find" text box, otherwise both upper case and lower case occurrences of the string will be found.
4. If you want the search string to be found as a whole word only and not as part of other words, select the "Whole Words Only" text box.
   If this box is left unselected, all occurrences of the string will be found.
5. If the search is being performed on an object listing, then the "Exclude collapsed blocks" check box is displayed. Select the check box to exclude collapsed blocks of source code from the search; leave blank to search the entire listing.
   For more information on expanding and collapsing object listings, see Expanding/Collapsing Object Listings.
6. In the "Direction" group frame, click the search direction up or down to specify whether the search will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
7. Choose **OK**.
   If no instance of the text searched for is found, a corresponding message is displayed.
   If an instance of the search string is found, it is displayed and selected.

To search for additional instances of the search string in the object

- From the **Edit** menu, choose **Find Next**.
  Or press **F3**.
  Alternatively, the text to find can be entered in the find combo box at the **Edit** toolbar.

Searching for and Replacing Source Text

▶ **To search for and replace a text string in the active program window**

1. From the **Edit** menu, choose **Replace**.
   Or click the **Replace Text** toolbar button.
   Or press **CTRL+H**.
   The "Replace" dialog box appears.
2. In the "Find" text box, enter the string to be searched for.
3. In the "Replace With" text box, enter the replacement string.
4. If you want the search to be case sensitive, select the "Case Sensitive" check box.
   The search will then only find the string exactly as it appears in the "Search For" text box, otherwise both upper case and lower case occurrences of the string will be found.
5. If the search string to be found as a whole word only and not as part of other words, select the "Whole Words Only" text box.
   If this box is left unselected, all occurrences of the string will be found.
6. If you want to confirm each change, leave the "Confirm Each Change" check box selected. If you deselect it, all instances of the search string will be converted to the replacement string without prompting you for confirmation.
7. In the "Direction" group frame, choose the search direction up or down to specify whether the search and replace will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
8. Choose **Replace**.
   If no instance of the text searched for is found, a corresponding message is displayed.
   If an instance of the search string is found, and the "Confirm Each Change" box is marked as in Step 6, a message box with the following pushbuttons is displayed:

Replace	Replace the search string with the replacement string and continue searching.
Skip	Skip to the next instance of the search string.
Cancel	Interrupt the search/replace operation.

Repeat Replace

If for any reason you interrupt the search/replace operation, you can resume it at any time using the Replace Next function.

1. To do so, from the **Edit** menu, choose **Replace Next**.
2. Or press **CTRL+F3**.

Searching for a Line Number

If you know which line you are searching for, you can use the "Go To" command to display a specific line number. In the function, you can specify whether you want to go to the numbered line or the physical line in the editor, depending on whether the line number option is switched on or off.

If you use the "Go To" command in a source listing, and the line specified is within a collapsed block, the block is expanded to show the desired line. For more information on expanding and collapsing object listings, see Expanding/Collapsing Object Listings.

▶ **To go to a specific line**

1. From the **Edit** menu, choose **Go To**.
   Or click the **Go To** toolbar button.
   Or press **CTRL+G**.
   The "Go To" dialog box appears.
2. If the line number option is turned on (program line numbers are displayed), select the **Numbered line** radio button. If the line number option is turned off (program line numbers are not displayed), select **Physical line**. The physical line number is the default.
3. Enter the line number to be found in the "Line Number" text box.
4. Choose **Go To**.
   The editor scrolls to the specified line and the cursor is placed at the beginning of the line.

Editing/Listing Referenced Natural Objects

While you are working in the program editor, you can open or list other Natural objects which are referenced in the program code, assuming these objects exist. If, for example, you are editing a program that calls a subprogram, you can open the subprogram, adapt it to the program, and return to the program.

To edit a referenced Natural object in your program

1. Click or double-click on the object.
2. From the **Program** menu, choose **Open Object**.
   Or press **CTRL+O**.
   The editor is opened.

To list the source code of a referenced Natural object in your program

1. Click or double-click on the object.
2. From the **Program** menu, choose **List Object**.
   The source code of the object is listed.

Splitting the Editor Window

Jumping between Split Screens

You can split the program editor window vertically or horizontally to view and modify two different parts of the object simultaneously. This feature saves you the trouble of printing a program to view two different sections simultaneously. Changes made in one section are made simultaneously in the other section.

To split the screen into two sections vertically or horizontally

1. From the **View** menu, choose **Vertical split** (**Horizontal split**).
   A vertical (horizontal) line appears in the middle of the window.
2. Split the screen into two sections by moving the mouse vertically (horizontally) to the position you want and clicking the left mouse button.
   The screen is split into two sections, each displaying the same information. You can now scroll each section individually and edit both sections as if they were part of the same object (as indeed they are).

To exit split-editor mode and return to a single screen

- From the **View** menu, choose **Unsplit**.
  Two editor sections are transformed into one.

Jumping between Split Screens

To jump between vertical/horizontal split screens

- Place the mouse pointer in the screen section of your choice and click.
  Or press **F6**.
  The cursor moves from one screen section to the other.

Expanding/Collapsing Object Listings

To improve the readability and maintainability of objects with complex program structures, you can display logical blocks of source code in expanded or collapsed form. Moreover, each individual logical block of code in an object can be expanded or collapsed, as required. Expandable/collapsible structures include, for example, DEFINE DATA blocks, REPEAT blocks, IF THEN ELSE blocks and READ blocks.

This feature applies only to source code listings of Natural objects other than DDMs which are saved in structured mode.

In the source code:

- square icons with minus signs (-) indicate the beginning of a collapsible block of code.
- square icons with plus signs (+) indicate the beginning of an expandable block of code.

The following topics are covered below:

- Making Listings Expandable/Collapsible
- Collapsing and Expanding Program Structures
- Additional Information

Making Listings Expandable/Collapsible

Object listings are not displayed by default in expandable/collapsible format. You must specify in the program editor that objects are to be listed in expandable/collapsible format. You can specify this on either session level or object level. Session-level settings are valid for all new listings in a Natural session; temporary settings are valid only for the active listing; they are lost once a listing is closed. If the listing is reopened, the session-level settings are used.

Expand/Collapse: Session Level

To set expandable/collapsible listings on a session-wide basis

1. From the **Object** menu, choose **New** and then, from the cascading menu, **Program**.
2. From the **Tools** menu, choose **Options**.
   Or press **Alt+ENTER**.
   In the "Options" dialog box select the tab "Program Editor"
3. Select the "Expand/Collapse" check box.
   The "Open collapsed" check box is activated.
4. Select the "Open collapsed" check box if listings are to be initially displayed in collapsed format. Leave empty to display the listings initially in expanded (normal) format.
5. Choose **OK**.

Collapsing and Expanding Program Structures

When a block of object code is collapsed, all of the lines of source code between the block begin statement and block end statement are hidden from view, including any other blocks if they are part of the chosen block. Hidden blocks retain their collapsed or expanded status.

► To collapse a program structure

- Double-click on the "-" icon marking the collapsible block of program code.
  Or place the cursor in the line containing the "-" icon and, from the **View** menu, choose Collapse **Block**.
  The block of code is collapsed.
  When a block of object code is expanded, all of the lines of source code between the block begin statement and block end statement are brought into view, including any other blocks if they are part of the chosen block. Previously hidden blocks retain the collapsed or expanded status they had before they were hidden.

► To expand a program structure

- Double-click on the "+" icon marking the collapsible block of program code.
  Or place the cursor in the line containing the "+" icon and, from the **View** menu, choose **Expand Block**.
  The block of code is expanded.

Additional Information

When searching a listed object, it is possible to exclude collapsed blocks from the search. For more information, see Searching for Source Text.

When you are searching for a line number in a listed object which is part of a collapsed block, the block is expanded to display the line. For more information, see Searching for a Line Number.

When you are splitting the editor window vertically or horizontally, one section can be displayed in collapsed format while the other is displayed in expanded format. From the **Options** menu, just choose **Expand/Collapse** for one of the window sections.

Recording/Replaying Keystrokes

You use the record/replay function to record and replay keystroke sequences which you want to repeat in the program editor. This is similar to a tape recorder with the functions record, stop, and play.

As the name implies, the keystroke recorder records and replays only input made directly from the keyboard, and not movements or selections made with the mouse. Thus, it is only possible to record within a single program editor window; recording in multiple program editor sessions is not possible.

To record a keystroke sequence

1. Place the cursor at the position in the editor where you want to begin recording.
2. From the **Tools** menu, choose **Start Recording**.
   Or press **Ctrl+Shift+R**.
3. Enter the key sequence you want to record.
   Only keyboard input is recorded. Mouse movements and operations are ignored.

To stop recording

- From the **Tools** menu, choose **Stop Recording**.
  Or press **Ctrl+Shift+S**.
  Recording ends.

To replay a recorded keystroke sequence

1. Place the cursor at the position in the editor where you want to begin replaying the recording.
2. From the **Tools** menu, choose **Replay Recording**.
   Or press **Ctrl+Shift+P**.
   The recording is replayed.

Using Context-Sensitive Help

- Syntax Help
- Syntax Coloring

Syntax Help

Within the Natural program editor, context-sensitive help is available for the following Natural syntax elements:

- Statements
- System variables
- System functions
- Parameters (for example, the AD parameter)

To call syntax help

- Place the cursor on a keyword within a syntax element for which you require help and press **F1**.
  Or double-click on a keyword within the syntax element and press **F1**.

Help for Statements

When you place the cursor on a keyword that forms part of a statement (for example, under the keyword TOP in the statement AT TOP OF PAGE), the help system tries to identify the statement that contains the keyword.

- If the result is unambiguous (that is, one complete statement is identified), help information is displayed for that particular statement.
- If a keyword is found which is used in several statements (for example, AT), a menu appears offering you a choice of statements containing the keyword you specified.
- If no complete statement can be found (for example, if you have only typed in the word DEFINE and pressed **F1** before typing out the full statement) or if the keyword is used in more than one statement, a dialog box appears offering you a choice of all statements beginning with or containing the keyword you specified.

Syntax Default Colors

Coloring is used in the program editor to mark various elements of syntax for better readability.

The default color assignments for Natural are:

- Blue - Natural keywords
- Red - Comments
- Green - Text constants, system functions, system variables
- Black - User-defined variables and remaining syntax elements

You can modify the color assignments. See Syntax Coloring.

Setting Editor Options

You can set preferences for various editor options. These settings are taken as default values each time you start the Program Editor.

To do so

- From the **Tools** menu, select **Options** and then in the options dialog select the tab "Program Editor".

Status Bar

Display the status line at the top of the editor window. For information on the status line contents, see Status Bar Information.

Line Numbers

Show the line numbers in the program editor.

Syntax coloring

Display color-coding for program syntax elements.

On how to modify color-coding, see Syntax Coloring.

Vertical scroll bar

Display the vertical scroll bar for the editor window.

Horizontal scroll bar

Display the horizontal scroll bar for the editor window.

Tabs

Specify the column numbers for tabulator stops in the program editor. You can add or modify existing tabulator stops in the text box.

Expand/Collapse

Enable logical blocks of code in object listings to be switched between a collapsed form and an expanded form. This is valid, for example, for DEFINE DATA blocks, REPEAT blocks, and IF THEN ELSE blocks. This option applies only to objects saved in structured mode. For more information on the expand/collapse functions, see Expanding/Collapsing Object Listings.

Open collapsed

Input to this check box is allowed only if the "Expand/Collapse" check box is marked. Initially display logical blocks of code in collapsed form on listing a program.

Max. number of actions

Specify the number of text operations which can be recalled/reinstated using the undo/redo functions. Zero signifies the maximum limit dependent only on the specified maximum memory size (see below). For more information on the undo/redo functions, see Undoing / Redoing Text Changes.

Max. memory size

Specify the amount of buffer space available for storing text operations made in the program editor. Zero signifies the maximum space available.

Alarm

Produce a "beep" sound when an invalid key or key combination is pressed.

Insert Alignment

Align the cursor with the first non-blank character of the previous line of text when you press **ENTER** or **RETURN**. If there are no non-blank characters before the inserted line, the cursor is aligned with column 1.

Renumber Before Save

Renumber the lines in a program and update line number references before every save.

Syntax Coloring

Coloring is used in the program editor to mark various elements of syntax for better readability.

You can define your own colors.

To do so

- From the "Program Editor Preferences" dialog, choose "Colors".
  The "Color Definition" window is displayed.
  You can define the colors of text type, as for example "Keywords", "System variables", and of the edtior window.

To define the color of the text type

1. Select the text type in the "Text Type" drop-down list box or click on it in the sample area.
   In the "Foreground" drop-down list box, select a color. (System is the color defined in Windows.)
2. In the "Background" drop-down list box, select a background color for the text. (System is the color defined in Windows.)
3. Choose **OK**.

The default color assignments for Natural are:

- Blue - Natural keywords
- Red - Comments
- Green - Text constants, system functions, system variables
- Black - User-defined variables and remaining syntax elements

To define the color of the Edit Window

1. Click anywhere in the sample area where there is no text.
   "Edit Window" is displayed in the "Text Type" drop-down list box.
2. In the "Background" drop-down list box, select a color. (System is the color defined in Windows.)
3. Choose **OK**.

Font Definition

You can define your preferred font in the "Windows Font" definition window.

To do so

- From the "Program Editor Preferences" dialog, choose "Font".
  The Windows "Font definition" window is displayed.
  For further information, please refer to the Windows documentation.

Note:
Only monospaced fonts are available.

Status Bar Information

The status line appears at the top of the window where the program is edited. It displays the following information:

Line: *x* of *y*

x: The current cursor line position.
y: The total number of lines in the object.

Col

The current cursor column position.

Size

The total number of characters in the source code.

Structured or Report

The programming mode (structured or report) of the active object.

You can change mode to structured with the system command "GLOBALS SM=ON" or to report "GLOBALS SM=OFF" or by selecting "Session Parameters" from the **Options** menu and then choosing **Miscellaneous** from the cascading menu.

Modified

Indicates that the object has been modified since the last save. If "Modified" is not displayed, then the object has not been modified since the last save.

INS or OVR

INS: Editor is in insert mode. Input does not overwrite existing text.
OVR: Editor is in overwrite mode. Input overwrites existing text.

Map Editor

You use the map editor to create and edit maps. After a map is created, you can store it in a library and invoke it using an INPUT USING MAP statement.

- Inserting Map Fields and Menus
- Modifying Map Contents
- Defining Fields
- Defining Field Attributes
- Defining an Array
- Modifying Field Colors and Representation
- Using Field Rules
- Defining Data Areas for Maps
- Testing Maps
- Previewing Maps
- Flipping Maps
- Modifying the Map Profile
- Setting Editor Options

Inserting Map Fields and Menus

A map consists of fields. You can create the following types of fields:

- text constants
- data fields
- menus
- push buttons
- bitmaps
- toggle buttons
- radio buttons
- selection boxes

Text fields and data fields correspond to the fields used in character-oriented Natural platforms (Mainframe Natural, for example). All other field types apply only to applications with graphical user interfaces.

> **Note:**
> Map editor field types are not the same as the dialog elements with the same name used in the dialog editor. Whereas the dialog editor’s dialog elements are identified by a handle definition in a data area, the map editor’s fields are not defined in a data area. Each is therefore addressed differently in Natural code: the map fields are addressed by an INPUT USING MAP statement, whereas the dialog elements are addressed by event-driven programming features.

The procedures in this section for inserting map fields assume the use of a mouse. The keyboard equivalents are provided in Keyboard Equivalents.

To insert a map field

1. From the **Insert** menu, choose a field type.
   Or click the toolbar button for the desired field type.
2. Move the mouse pointer into the editor.
   The cross-hair pointer is displayed with a symbol for the selected field.
3. Position the mouse pointer at the place in the map where you want to put the field, hold down the left mouse

button and drag the mouse to size the field (up, down, right, or left, depending on the type of field).
A box is displayed to indicate graphically the size of the field you are creating. Its actual length is displayed in the "Len" field in the editor status line.

4. Release the mouse button.
The map field appears in the map editor. It is selected and its properties are displayed in the status line.

To create a map menu field using the mouse

- From the **Insert** menu, choose **Menu**.
Or click the **Create Menu** toolbar button.
If no menu has been created yet for the map, a menu bar appears at the top of the map editor with a menu field. The menu field is selected.
If a menu has already been created for the map, then a menu field is added to the menu bar. The menu field is selected.

For information about how to manipulate map fields once they have been created, see Modifying Map Contents.

For information about how to import variables from other Natural objects, see Importing Fields.

Modifying Map Contents

- Selecting Fields
- Deselecting Fields
- Copying Fields
- Cutting Fields
- Pasting Fields
- Deleting Fields
- Moving Fields
- Resizing Fields
- Aligning Fields
- Importing Fields
- Importing System Variables
- Keyboard Equivalents

Selecting Fields

To select a single field

- Point to the field and click.
  The field handles appear.

To select more than one field

1. Point to a spot outside the range of fields to be selected.
2. Drag the mouse across the map, drawing a rubber band box that surrounds the fields.
3. Release the mouse button and the field handles appear.

or

1. Select the fields requested by pressing the **SHIFT** key and the left mouse button together.
2. To move the fields drag the selected area to where needed, by keeping the left mouse button pressed.
3. Release the mouse button to place the fields.

Deselecting Fields

To deselect a field

- Move the pointer away from the field and click.
  The field handles disappear.

Copying Fields

Fields can be copied within the same map or between two different maps.

To copy a field

1. Select the field(s) to be copied using the instructions provided in Selecting Fields.
2. From the **Edit** menu, choose **Copy**.
   Or click the **Copy**toolbar button.
   Or press **CTRL+C**.
   The field is copied to the clipboard and can now be pasted within the same map or another map. For instructions on pasting fields, see Pasting Fields.

Cutting Fields

The cut function can be used to delete fields from a map or to move fields within/between maps. When a field is cut, it is taken from the map and placed on the clipboard. It remains there until the next cut or copy operation is performed, at which time it is irretrievably discarded from the clipboard to make way for the next cut/copied field.

To cut a field

1. Select the field(s) to be cut using the instructions provided in Selecting Fields.
2. From the **Edit** menu, choose **Cut**.
   Or click the **Cut** toolbar button.
   Or press **CTRL+X**.
   The field is cut to the clipboard and can now be pasted within the map or to another map. For instructions on pasting fields, see Pasting Fields.

Pasting Fields

The paste function is used to place a field at a specific position within an editor after it has been copied or cut to the clipboard from another position within the same map or another map. A field which has been copied or cut to the clipboard can be pasted repeatedly without recopying it.

To paste a field

1. Copy or cut a field as described in Copying Fields or Cutting Fields.
2. If the field is to be pasted in another map, select the map.
3. From the **Edit** menu, choose **Paste**.
   Or click the **Paste**toolbar button.
   Or press **CTRL+V**.
   The field is pasted to the map.
4. To paste the same field again, repeat Steps 2 and 3.

Deleting Fields

When a field is deleted, it is cut from the map but is *not* placed on the clipboard.

▶ **To delete a field or a range of fields**

1. Select the field(s) to be deleted using the instructions provided in Selecting Fields.
2. Select the field or range of fields.
3. From the **Edit** menu, choose **Delete.**
   Or click the **Delete** toolbar button.
   Or press **DEL**.
   The field is deleted from the map.

Moving Fields

▶ **To move a field or a range of fields to a different location on the map**

1. Select the field(s) to be moved using the instructions provided in Selecting Fields.
2. Place the pointer within the field handles and drag the field or range of fields to the new location.
3. Release the mouse button.

Resizing Fields

▶ **To resize a field**

1. Select the field(s) to be resized using the instructions provided in Selecting Fields.
2. Point to any of the field handles surrounding the field.
   The pointer changes to a double-sided arrow.
3. Drag the mouse until the field(s) reach the desired length.
4. Release the mouse button.

Aligning Fields

As an alternative to arranging fields within a map individually using the move function, you can arrange fields more accurately with respect to each other or with respect to the map by using the align function. You can align fields in a map in the following ways:

- Justify selected fields to the left, right, top, or bottom of their field handles.
- Center selected fields vertically or horizontally with respect to each other.
- Center selected fields horizontally with respect to the map editor window.

▶ **To align fields**

1. Select the field(s) to be aligned using the instructions provided in Selecting Fields.
2. From the **Field** menu, choose **Alignment** and, from the cascading menu, one of the entries.
   Or click on one of the alignment toolbar buttons.
   The selected fields are aligned.

Importing Fields

You can import data fields, system variables, toggle buttons, selection boxes, and radio buttons into the active map. Fields can be imported from any object, including DDMs, in any library. Imported fields are placed on the system clipboard. You can paste them into as many map editor windows as desired.

To import one or more fields from another object into the map editor window

1. From the **Insert** menu, choose **Import**.
2. From the cascading menu, choose the type of field you want to import, choose **Data Field**, **Toggle Button**, **Selection Box** or **Radio Button**.
   The "Import" dialog box appears. The name of the current library is displayed in the "Library" list box.
3. If the object containing the fields you want to import is located in a different library, open the "Library" list box and select the library.
4. From the "Type" group frame, select the type of Natural object from which you want to import fields.
   A list of all Natural objects in the current library of the type you selected appears in the "Object List" box.
5. Select the object that contains the fields you want to import.
   The fields in the selected object appear in the "Data Fields" list box.
6. Select the fields that you want to import.
7. Choose **Import**.
8. Choose **Quit**.
   The dialog box closes and the fields appear in the upper left corner of the map editor window. You can move the fields around within the map.

Importing System Variables

To import one or more system variables into the map editor window

1. From the **Insert** menu, choose **Import**.
2. From the cascading menu, choose **System Variable**.
   The "Import System Variable" dialog box appears.
3. Select the system variables that you want to import.
4. Choose **Import**.
5. Choose **Quit**.
   The dialog box closes and the system variables appear in the upper left corner of the map editor window. You can move the system variables around within the map editor window.

Keyboard Equivalents

Most of the actions described in this section can be performed using the keyboard instead of the mouse. The table below provides key sequences for each action.

This action	Is performed with this keyboard action
Move the mouse pointer	Press arrow keys.
Select map field	Place mouse pointer on field and press SPACEBAR.
Select map fields	Place mouse pointer outside field, press and hold down SPACEBAR, press arrow keys to encircle map fields to be selected, release SPACEBAR.
Deselect map field(s)	Move mouse pointer outside field(s) and press SPACEBAR.
Move map field(s)	Select map field(s), press and hold down SPACEBAR and press arrow keys.
Copy map field	Select map field and press CTRL+C
Cut map field	Select map field and press CTRL+X
Paste map field	Select map field and press CTRL+V
Delete map field	Select map field and press Del
Resize map field	Select map field, move mouse pointer to a field handle, press and hold down SPACEBAR and press arrow keys.
Local Data	CTRL+ALT+L
Parameter Data	CTRL+ALT+P

Defining Fields

When a map field is inserted, it is given a default field definition and/or default field attributes which can be modified at any time.

To modify a definition for a field

1. Select a field.
2. From the **Field** menu, choose **Definition**.
   The **Field Definition** dialog appears.

For detailed instructions, see the following sections:

- Defining Text Fields
- Defining Data Fields
- Defining Selection Boxes
- Defining a Radio Button
- Defining a Toggle Button
- Defining Menu Items
- Defining Push Buttons
- Defining Bitmaps

Defining Text Fields

is a constant whose format is always alphanumeric. You can modify the text in a text field at any time.

To modify the text field

1. Double-click the text field.
   Or, from the **Field** menu, select the text field and choose **Definition**.
   The background in the text field is highlighted.
2. Enter the desired text directly in the text field. For text field modification, you can also use the context menu.
   If the text is long, it could be necessary to increase the size of the text field. See Resizing Fields.
3. Deselect the text field.

Defining Data Fields

To modify a data field definition

1. Double-click the field.
   Or, from the **Field** menu, select the field and choose **Definition**.
   The "Field Definition" dialog box appears.
2. The "Field" text box contains the current name of the field. You can change it by typing in a new name.
3. In the "Format" list box, choose a format for the field. The default format is alphanumeric.
   Note:
   Because the information required to define a field depends on its format, text boxes in the "Field Definition" dialog box may appear/disappear when the format changes.
   For a list of valid formats, see **Definition of Format and Length** in the Natural Reference documentation.
4. In the "Length" list box, enter the desired length of the field.
   This field is only displayed for data formats F, A, B, I, N and P.
5. In the "DF" list box, enter the desired date format.
   This field is only displayed for data format D.
6. In the "AL" text box, enter the desired output length for the alphanumeric field. This can be longer or shorter than the actual field.
   This field is only displayed for data format A.
7. In the "NL" text box, enter the desired output length for the numeric field. This can be longer or shorter than the actual field.
   This field is only displayed for data formats B, I, N and P.
8. In the "FL" text box, enter the desired floating-point mantissa length during input or output. The total length is FL+6 for sign, exponent, and decimal character.
   This field is only displayed for data format F.
9. Select the "SG" toggle button to specify whether a sign position is to be allowed for the field.
   This field is only displayed for data formats F, I, N and P.
10. **Rules:** The "Rules" field displays the number of processing rules that are defined for the data field.
11. **Mode:** The "Mode" field displays the current mode of the data field. The following table describes the possible modes.

Data	The field was created by selecting a field from a DEFINE DATA definition.
Sys	The field is a system variable.
Undef	The field was created directly on the screen and has a dummy name.
User	The name of the field was created by changing the field name
View	The field was created by selecting a field from a view.

12. Select the **Array** toggle button to define an array for the data field.
    The **Array**button is enabled. For more information, see Defining an Array.
13. **AD:** The "AD" field displays the current attribute definition for the data field. For instructions on how to modify the attribute definitions, see Defining Field Attributes.

14. From the "PM" list box, choose a print mode for the field.
    The following print modes are available:

blank	No print mode (default)
C	An alternative character set is used.
I	Inverse print direction
N	Normal print direction

15. In the "CD" list box, choose a color definition for the field content.
    Note:
    You can also define a color for a field by choosing **Color** in the **Field** menu.
16. In the "CV" text box, you can enter a dynamic field attribute control variable.
    This is the control variable that will contain the attributes to be used for the data field. The variable must be defined with format C (for attribute control) in the program that references the map.
    The control variable also contains a MODIFIED data tag, which indicates if the field has been modified following map execution. A single control variable can be applied to several map fields, in which case the MODIFIED data tag will be set if any of the fields referencing the control variable have been modified.
    See the Natural Reference documentation for more information on the CV parameter.
17. In the "Dim" list box, you can specify the number of dimensions for an array of control variable. The default is none.
    You must mark the array box in order to specify the control variable as an array.
18. In the "DY" text box, enter the dynamic string attributes.
    The dynamic string parameter is used to assign attributes for dynamic attribute field display. See the Natural Reference documentation for more information on the DY parameter.
19. Select the **ZP** toggle button to specify zero printing for the field. If this button is selected, zero values are printed as one zero. If this button is not selected, zero values are suppressed.
    This field is only displayed for data formats F, I, N and P.
20. In the "EM" text box, enter the edit mask to be used for the data field.
    See the Natural Reference documentation for more information on the EM parameter.
    Note:
    The display length is overridden by the edit mask.
21. In the "Help Routine" text box, enter the name of the help routine that is to be invoked for the data field.
    Specify a text constant or a user-defined variable that contains the name of the routine.
22. In the "Help Parameters" text box, enter the name of the parameter that is to be passed to the help routine.
    Specify a constant or a user-defined variable that contains the value of the parameter.
    If an "=" is specified, the name of the field as defined in the map definition is passed to the help routine.
    When a help routine is assigned to a map, "=" denotes the name of the map.
    Since no explicit DEFINE PARAMETER statement can be specified with the map editor, the format and length of "Help Parameters=" are defined in the following way:
    If the value specified for operand2 in the HE (Helproutine) session parameter is defined as a map field, the format/length definition of this map field is used to define the format and length of "Help" Parameters. If not, the parameter specified for "Help" Parameters must be defined as N7 (default format assumed) in the program using the map.
23. If you want to use "Help" Parameters as an array, select the **Array** toggle button and choose the **Array** button.

For additional information, see Defining an Array.

Defining Selection Boxes

To define a selection box, use the same procedure as described under Defining Data Fields.

To define attributes for a selection box, see Defining Field Attributes.

Defining Selection Box Items

To define items for a selection box

1. Access the "Field Definition" dialog for the selection box.
2. Choose "Items".
   The "Selection Box Definition" dialog appears. The field name is displayed in the "Selection Box" field.
   The existing items are displayed in the "Items" list box. A default item "Item" is displayed.
   From this dialog box, you can perform the actions described in the following sections.

Defining Constant Selection Box Items

To add an item constant

1. Choose **Add Constant**.
   The "Selection Box Item Constant" dialog box appears.
2. In the "Constant" text field, enter the name of the new item.
   The length of the constant cannot be longer than the value you specified in the "Length" list box of the "Field Definition" dialog.
3. Choose **OK** to define the selection box item.

Defining Variable Selection Box Items

To add an item variable

1. Choose **Add Variable**.
   The "Selection Box Item Variable" dialog box appears.
2. In the "Variable" text box, enter the name of the variable to serve as an item.
   The variable must be a valid Natural identifier. The length of the variable value is fixed at the value set for "Length" in the "Field Definition" dialog.
3. If the variable is an array, you can define the array by choosing **Define Array**.
4. Choose **OK** to define the selection box item.

Box Items

▶ **To import an item**

1. Choose **Import Item**.
   The "Import Selection Box Item" dialog box appears. The name of the current library is displayed in the "Library" list box.
2. If the object containing the fields you want to import is located in a different library, open the "Library" list box and select the library.
3. From the "Type" group frame, select the type of Natural object from which you want to import fields.
   A list of all Natural objects in the current library of the type you selected appears in the "Object" List box.
4. Select the object that contains the fields you want to import.
   The fields in the selected object appear in the "Data Fields" list box.
5. Select the fields that you want to import.
6. Choose **Import**.
   The dialog box closes and the fields appear at the bottom of the "Items" box.

Modifying a Selection Box Item

▶ **To modify an item**

- In the "Items" list box, select the item and choose **Modify Item**.
  If the item is a constant, then the "Selection Box Item Constant" dialog box appears. See Defining Constant Selection Box Items.
  If the item is a variable, the "Selection Box Item Variable" dialog box appears. See Defining Variable Selection Box Items.

Removing a Selection Box Item

▶ **To delete an item**

- In the "Items" list box, select the item and choose **Remove Item**.
  The item is removed from the "Items" list box.

Moving Selection Box Items

When you add an item to a selection box, it is placed at the bottom of the "Items" list box. In most cases, you will want to reorder these items in logical groupings.

▶ **To move an item to another position in the list box**

1. Select the selection box item and drag the cursor to the new position
   A dashed line between items indicates the position to which the item will be moved.
2. Drop the item in the new position.
   The item is inserted at the new position.

Defining a Radio Button

To define a radio button, use the same procedure as described under Defining Data Fields.

To define attributes for a radio button, see Defining Field Attributes.

Defining Radio Button Contents

To define the contents for a radio button

1. Access the "Field Definition" dialog for the radio button.
2. Choose **Contents**.
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the radio button is to be a constant or a variable.
4. In the "Name" text box, enter the name of the constant or variable. If you want to import an alphanumeric field from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
   The length of a variable definition cannot exceed the length of the radio group definition.
5. If the contents are an array, then select the **Array** toggle button and choose "Define Array".
   For information on how to define an array, see Defining an Array.
6. Choose **OK** to complete definition.

Defining a Toggle Button

To define a toggle button, use the same procedure as described under Defining Data Fields.

To define attributes for a toggle, see Defining Field Attributes.

Defining a Toggle Button Label

The format is always L (Logical).

To define the label for a toggle button

1. Access the "Field Definition" dialog for the toggle button.
2. Choose **Label**.
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the toggle button is to be a constant or a variable.
4. In the "Name" text box, enter the name of the constant or variable. If you want to import an alphanumeric field from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
5. In the "Format" list box, select the format for the variable.
6. In the "Length" text box, enter the length of the variable.
7. If the contents are an array, then select the **Array** toggle button and choose **Define Array**.
   Note: The start values for the toggle button labels are set to 1 as default. To change the start value of one label, select the element to be changed and set the new start value.
   For information on how to define an array, see Defining an Array.
8. Choose **OK** to complete definition.

Defining Menu Items

To define items for a menu

- In the menu bar, double-click any menu item.
  The "Edit Menu" dialog box appears. From this dialog box, you can perform the following actions:

Editing a Menu Name

▶ **To edit a menu name**

1. From the "Edit Menu" dialog box, choose **Menu Name**.
   The "Edit Constant or Variable" dialog box appears.
2. In the "Type" group frame, specify whether the menu name is to be a constant or a variable.
3. In the "Name" text box, modify the name of the constant or variable. If you want to import a variable from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
4. If the menu is an array, then select the **Array** toggle button and choose **Define Array**.
   For information on how to define an array, see Defining an Array.
5. Choose **OK** to complete definition.

Adding a Menu Item

▶ **To add items to a menu**

1. From the "Edit Menu" dialog box, choose **Add Item**.
   The "Define Menu Item" dialog box appears.
2. Choose **Item Name.**
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the menu item is to be a constant or a variable.
4. In the "Name" text box, modify the name of the constant or variable. If you want to import a variable from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
5. If the item is an array, then select the **Array** toggle button and choose **Define Array**.
   For information on how to define an array, see Defining an Array.
6. Choose **OK**.
   The "Define Menu Item" dialog box appears.
7. From the "Key Name" list box, choose a PF key name for the item.
   PF keys which have already been allocated do not appear in the list box.
8. Select the **Enabled** toggle button to permit menu item selection. Otherwise, the menu item cannot be selected.
9. Choose **OK**.
   The "Edit Menu" dialog is displayed with the new menu item.

Adding a Submenu

To define a submenu for a menu item

1. From the "Edit Menu" dialog box, choose **Add Submenu**.
   The "Edit Submenu" dialog box appears.
2. From the "Edit Submenu" dialog box, choose **Submenu Name**.
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the submenu name is to be a constant or a variable.
4. In the "Name" text box, modify the name of the constant or variable. If you want to import a variable from another Natural object, select **Variable** > **Import**.
   For information on how to import fields from other Natural objects, see Importing Fields.
5. If the submenu is an array, then select the **Array** toggle button and choose **Define**.
   For information on how to define an array, see Defining an Array.
6. Choose **OK**.
   The "Edit Submenu" dialog box appears.
7. To add items to the submenu, choose **Add Item** and follow the instructions under Adding a Menu Item.
8. To add a separator to the submenu, choose **Add Separator** and follow the instructions under Adding a Menu Separator.
9. To modify a submenu item, choose **Modify Item** and follow the instructions under Modifying a Menu Item.
10. To remove a submenu item, choose **Remove Item** and follow the instructions under Removing a Menu Item.
11. Choose **OK** to complete the submenu definition/modification.

Adding a Menu Separator

You can designate logical groupings in your menus or submenus by separating these groupings with horizontal lines.

To add a separator to a menu or submenu

1. From the "Edit Menu" or "Edit Submenu" dialog box, choose **Add Separator**.
   A selected separator is placed behind the last menu item.
2. Drag the separator to the new position.
   A dashed line between items indicates the position to which the separator will be moved.
3. Drop the separator in the new position.
   The separator is inserted at the new position.

Modifying a Menu Item

To modify a menu item

1. In the "Menu Items" list box, select the item and choose **Modify Item**.
   The "Define Menu Item" dialog box appears.
2. Follow the instructions as described in Adding a Menu Item.

Removing a Menu Item

▶ **To delete a menu item**

- In the "Menu Items" list box, select the menu item and choose **Remove Item**.
  The menu item is removed from the "Menu Items" list box.

Moving a Menu Item

When you add an item to a menu or submenu, it is placed at the bottom of the "Menu Items" list box. In most cases, you will want to reorder these items in logical groupings.

▶ **To move an item to another position in the list box**

1. Select the menu item and drag the cursor to the new position
   A dashed line between items indicates the position to which the item will be moved.
2. Drop the item in the new position.
   The item is inserted at the new position.

Defining Push Buttons

▶ **To define a push button**

1. Double-click the push button.
   The "Define Push Button" dialog box appears.
2. Choose **Edit Label**.
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the push button name is to be a constant or a variable.
4. In the "Name" text box, modify the name of the constant or variable. If you want to import a variable from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
5. If the push button is an array, then select the **Array** toggle button and choose **Define Array**.
   For information on how to define an array, see Defining an Array.
6. Choose **OK**.
   The "Define Push Button" dialog box appears.
7. From the "Key Name" list box, choose a PF key name for the push button.
   PF keys which have already been allocated do not appear in the list box.
8. Select the **Enabled** toggle button to permit push button selection. Otherwise, the push button cannot be selected.
9. Choose **OK**.
   The push button is displayed with the new name.

Defining Bitmaps

To define a bitmap

1. Double-click the bitmap.
   The "Define Bitmap" dialog box appears.
2. Choose **Edit Label**.
   The "Edit Constant or Variable" dialog box appears.
3. In the "Type" group frame, specify whether the bitmap name is to be a constant or a variable.
4. In the "Name" text box, modify the name of the constant or variable. If you want to import a variable from another Natural object, select **Variable** > **Import**.
   For information on how to import variables from other Natural objects, see Importing Fields.
5. If the bitmap is an array, then select the **Array** toggle button and choose **Define Array**.
   For information on how to define an array, see Defining an Array.
6. Choose **OK**.
   The "Define Bitmap" dialog box appears.
7. Choose **Edit File Name**.
   The "Edit Constant or Variable" dialog box appears.
8. Repeat Steps 3 to 6.
9. From the "Key Name" list box, choose a PF key name for the bitmap.
   PF keys which have already been allocated do not appear in the list box.
10. Select the **Enabled** toggle button to permit bitmap selection. Otherwise, the bitmap cannot be selected.
11. Select the **Scale Bitmap** toggle button to cause the bitmap to be displayed with the size defined by the map. Otherwise, the bitmap is displayed in its original size.
12. Choose **OK**.
    The bitmap is displayed with the new name.

Defining Field Attributes

Field attributes can be defined for data fields, toggle buttons, radio buttons and selection boxes.

To define the attributes of a field

1. In the "Field Definition" dialog box for the appropriate field, choose **Attributes**.
   The "Attribute Definitions" dialog box appears.
2. To change the definition for an attribute, open the list box for the attribute and select a different value. A description of the options for each attribute is provided later in this section.
3. To change the filler character, in the "Filler Character" text box, enter a different character.
   Empty input fields or modifiable output fields are filled with this character. When you edit an input field which is padded with blanks to its maximum length, it may appear as if the text cannot be edited. In such cases, the blanks must be explicitly deleted from the end of the field.
4. Choose **OK**.

Attribute	Code	Explanation
Field Representation Attributes		
Blinking	**B**	Value is displayed blinking.
Cursive/Italic	**C**	Value is displayed cursive/italic.
Default	**D**	Value is displayed with normal intensity.
Intensified	**I**	Value is displayed intensified.
Non-display	**N**	A value entered in the field will not be displayed.
Underlined	**U**	Value is displayed underlined.
Reverse video	**V**	Value is displayed in reverse video.
Dynamic attributes	**Y**	Attributes are controlled via a control variable.
Field Alignment Attributes		
Left-justified	**L**	Value is displayed left-justified.
Right-justified	**R**	Value is displayed right-justified.
Leading zeros right-justified.	**Z**	Numeric values are displayed with leading zeros,
Field Input/Output Characteristics		
Input field, non-protected	**A**	Field is an input field and non-protected.
Output field, modifiable	**M**	Field is an output field and can be modified.
Output field, protected	**O**	Field is an output field and write-protected.
Temporarily protected	**P**	Used in conjunction with control variable.
Mandatory Input Characteristics		
Value mandatory	**E**	Value must be entered in the field. Only relevant for input-only fields (AD=A).
Value optional	**F**	Value can, but need not, be entered in the field.
Length of Input Value Characteristics		
Fixed input length	**G**	Value entered in field must have same length as field. Only relevant for input-only fields.
Variable input length	**H**	Value entered in field can be shorter than field.
Field Upper/Lower Case Characteristics		
Translate lower to upper	**T**	The value entered is translated to upper case.
Accept lower case	**W**	Lower case values are to be accepted.

Defining an Array

You can define an array of up to three dimensions for a data field. The order in which the dimensions of the array are mapped to the map layout is determined by the values you enter.

The "Define Array" push button is inactive unless the "Array" toggle button is selected.

To define an Array and its dimensions for the data field

1. Choose **Array**.
   The "Define Map Array" dialog box appears.
2. From the "No. Dimensions" list box, select the number of dimensions for the array.
   The default number of dimensions is 1. Text boxes appear for the number of dimensions specified.
3. In the "Line Spacing" text box, enter the number of blank lines to be inserted horizontally between each dimension occurrence in the array.
4. In the "Column Spacing" text box, enter the number of blank columns to be inserted vertically between each dimension occurrence in the array.
5. For a field defined as a three-dimensional array the "CLS/LS" text box is displayed. Enter the number of blank lines to be inserted vertically or columns to be inserted horizontally between dimensions 1, 2, or 3, depending on the layout definition for dimension 3.
   The values specified in the Layout list boxes determin in which direction each of the dimensions are painted in the edit area of the map. (V = vertical, H = horizontal, VD/HD = vertical/horizontal, position of third dimension)
6. In the "Start From" text box, enter the starting index value for each dimension.
   You can enter a number or the name of a variable; the actual value is supplied in the program that invokes the map definition. Unless defined otherwise as a field in the map, the variable is assumed to be defined as format/length N7.
7. In the "Upper Bounds" text box, enter a value for each dimension.
   This number is the highest occurrence of the first, second and third dimension. A field defined in a program (a user-defined variable or a database field) can be imported to define the map array. In this case, the upper bounds of the field, as defined in the program, are used. All values can be overwritten for imported data fields. However, when you choose **OK** in the "Field Definition" dialog, you are asked if you really want to change an imported field’s definition.
   If the array is initially defined in the map, the upper bounds are set by the number of occurrences defined for the vertical, horizontal and fixed indexes.
8. In the "Occurrences" text box, enter a value for each dimension.
   This is the number of occurrences that will be displayed on the map. This does not apply to the third dimension of a 3-dimensional array because only two ranges of occurrences can be displayed on the screen. One-dimensional arrays can be displayed as multi-line/multi-column fields. For these arrays, the second number of occurrences and the layout for the second dimension can be defined.
9. From the "Layout" list box, select a layout value for each dimension.
   Layout values determine the axis assumed by each dimension of the array. This determines how the array is represented in the map layout. For arrays of one and two dimensions, the only possible values are H (horizontal axis) and V (vertical axis). For arrays with three dimensions, you must decide how the third dimension will be represented. The options are HD (horizontally detached) or VD (vertically detached). The dimension specified as HD or VD is represented by members which are grouped in a horizontal or vertical orientation.
10. Choose **OK**.
    The array is defined and you are returned to the "Field Definition" dialog box.

Splitting arrays into multiple parts is currently not supported by the map editor (that is, there cannot be two arrays with the same name and non-overlapping bounds, as is possible in Natural for mainframe platforms). Interleaving arrays are supported.

When defining multi-dimensional arrays, note that the defaults for the first and second dimensions have been reversed, which means that horizontal will be vertical and vice versa; in addition, the first and second dimensions can now also be specified as "fixed"; the third dimension is still "fixed" by default.

Changing the Number of Displayed Occurrences in an Array

Once you have defined an array, you can use the mouse to graphically modify the number of displayed occurrences and, indirectly, the upper bounds without accessing the "Define Map Array" dialog box.

To modify the number of displayed occurrences and/or upper bounds for an array

1. Place the mouse pointer on one of the four corner field handles.
2. Drag the mouse to increase/decrease the number of elements in the array.
   During resizing, a dotted outline indicates the intended modification.
3. When you have reached the desired size for the array, release the mouse button.
   If you increase the number of displayed occurrences for a dimension beyond the upper bound, then the upper bound is automatically increased as well. If you decrease the number of displayed occurrences for a dimension, the upper bound remains at its previous value.

Modifying Field Colors and Representation

For any text constant, data field, toggle button, radio button or selection box you can specify a color and a display style for its name.

To define the color and/or representation for a field

1. In the map editor, select the field.
2. From the **Field** menu, choose **Color**.
   Or click the **Color/Representation** toolbar button.
   The "Field Color and Representation" dialog box appears.
3. From the "Color Selection" group, select a color.
4. From the "Field Representation" group, select an option.
5. Choose **OK**.

Using Field Rules

- Creating Field Rules
- Copying Field Rules
- Editing Field Rules
- Changing Field Rule Rank
- Unlinking (Deleting) Field Rules
- Defining Free Predict Rules
- Converting Free Rules to Inline Rules
- Defining Key Rules

Field rules can be defined for any data field, toggle button, radio button or selection box. A field can have up to 100 processing rules (ranked 0-99). At map execution time, the processing rules are executed in ascending order, first by rank, then by screen position of the field.

For optimum performance, the following assignments are recommended when assigning ranks to processing rules:

Rank	Processing Rule
0	Termination rule
1-4	Automatic rules
5-24	Format checking
25-44	Value checking for individual fields
45-64	Value cross-checking between fields
65-84	Database access
85-99	Special purpose

Processing rules can be defined as either inline processing rules or free Predict rules.

Inline processing rules are rules that are defined within a map source and do not have a name assigned. An ampersand (&) within the source code of a processing rule will be dynamically substituted with the fully qualified name of the field using the rule. For example:

```
IF & = ' ' REINPUT 'ENTER NAME' MARK *&
```

Note:
To be able to access Predict on a UNIX or mainframe server, you must have set up your NATPARM parameter file accordingly and established a corresponding link by using Natural RPC. For information on how to assign dictionary servers and how to use Natural RPC, refer to the Natural Remote Procedure Call (RPC) documentation.

Creating Field Rules

To create a field rule

1. Select a map field.
2. From the **Field** menu, choose **Rules**.
   The "Field Rule" dialog box appears.
3. From the "Field Rule" dialog box, choose **Create**.
   A program editor window appears.
4. Enter the rule.
5. From the **Object** menu, choose **Save** to save the rule.
   The "Rule Selection" dialog box appears.
6. In the list box, select a rank number.
   The selected rank appears in the "Rank " box.
7. Choose **OK**.
   At this point, you can create another new rule or make a copy of the new rule and save it with a different rule number.

Copying Field Rules

To copy a field rule

1. Select the map field containing the field rule to be copied.
2. From the **Field** menu, choose **Rules**.
   The "Field Rules" dialog box appears.
3. In the "Ranks" list box, select the number of the rule to be copied.
4. Choose **Copy**.
   The "Rule Selection" dialog box appears.
5. In the "Rank" list box, select a rank for the new rule.
6. Choose **OK**.
   If you chose the rank number of an existing rule, then you are asked whether you want to overwrite the rule. Otherwise, the rule is copied and the "Field Rules" dialog box is redisplayed. The rule is displayed in the "Rule Text Fragment" box.

Editing Field Rules

To edit a field rule

1. Select the map field containing the field rule to be edited.
2. From the **Field** menu, choose **Rules**.
   The "Field Rules" dialog box appears.
3. In the "Ranks" list box, select the rank number of the rule to be edited.
4. Choose **Edit**.
   The program editor appears.
5. Edit the rule and then, from the **Object** menu, choose **Save** to save the rule.
6. Close the program editor.

Changing Field Rule Rank

To change the rank of a field rule

1. Select the map field containing the field rule to be modified.
2. From the **Field** menu, choose **Rules**.
   The "Field Rule" dialog box appears.
3. In the "Ranks" list box, select the rank number of the rule to be reranked.
4. Choose **Move**.
   The "Rule Selection" dialog box appears.
5. In the "Rank" list box, select another rank for the rule.
6. Choose **OK**.

Unlinking or Deleting Field Rules

Unlinking is the same as deleting a field rule.

To delete a field rule

1. Select the map field containing the field rule to be deleted.
2. From the **Field** menu, choose **Rules**.
   The "Field Rule" dialog box appears.
3. In the "Ranks" list box, select the rank number of the rule to be deleted.
4. Choose **Unlink**.
   You are asked if you really want to unlink (delete) the field rule.
5. Choose **Yes** to delete the field rule.

Defining Free Predict Rules

▶ To define a free rule; that is, to link a free rule to a selected field

1. Select a map field.
2. From the **Field** menu, choose **Rules**.
   The "Field Rule" dialog box appears.
3. Choose **Free Rule**.
   Since free rules cannot be created but only selected, the **Create** push button changes to "Select".
4. Choose **Select**.
   The "Free Rule Selection" dialog box appears.
5. Enter the rule name, the Predict owner and up to five keywords under which the rule is stored in Predict.
   Asterisk notation can be used for the rule name, the keywords can be combined with the boolean operators AND and OR, and a "BUT NOT" keyword can be specified, too.
   A free rule selection list is displayed, from which you can select the rule(s) you want to link to the field.
   Choose **Read Source** to read the source code into the "Free Rule Text Fragment information" box of a selected rule.
6. Choose **OK**.
   With each selected rule, you are asked to assign a rank.
7. Specify a rule rank.
   The rule is now linked to the field. At this point, you can convert a linked free rule into an inline rule as described in Converting Free Rules to Inline Rules.

Converting Free Rules to Inline Rules

▶ To convert a linked free Predict rule to an inline rule

1. From the **Field** menu, choose **Rules**.
   The "Field Rule" dialog box appears.
2. Select the rank of the free rule you want to convert.
   The "Free Rule" toggle button is selected and the Free Rule appears in the "Rule Text Fragment information" box.
3. Deselect the **Free Rule** toggle button.
   A message box appears asking you whether you want to change the rule type to inline.
4. Choose **OK**.
   The free rule becomes an inline rule and its source code is displayed in the "Rule Text Fragment information" box.

Defining Key Rules

The PF Key Rules function allows you to create, edit, move, copy and unlink (delete) function key related processing rules for the active map. Key rules also can be defined as either inline processing rules or free Predict rules.

Key rules can be used to assign activities to program sensitive function keys during map processing. For function keys that already have a command assigned by the program, this command is executed without any rule processing.

Example:

```
IF *PF-KEY = ’PF3’ ESCAPE ROUTINE END-IF
```

When this rule is executed, map processing is terminated without further rule processing.

To create, copy, and edit function key rules

- From the **Map** menu, choose **Pfkey rules**.
  The "PF-Key Rules" dialog box appears.
  Key rules are defined exactly like field rules. For detailed instructions, see Using Field Rules.

Defining Data Areas for Maps

You can define a local data area for a map to define variables to be used for map processing rules. You can define a parameter data area for a map to define the parameters that can be received from programs.

The procedures for defining, modifying, and deleting local and parameter data elements are virtually identical and are therefore explained generically below.

- Defining a Data Element
- Modifying a Data Element
- Removing a Data Element

Defining a Data Element

▶ **To define a data element for a map**

1. From the **Map** menu, choose **Local Data** or **Parameter Data**.
   The "Define Local/Parameter Data" dialog box appears.
2. Choose **Add** to add a data definition.
   The "Data Definition" dialog box appears.
3. In the "Name" text box, enter the name of the data element to be defined.
4. From the "Format" drop-down list box, select a data format.
5. If you selected the format A, B, F, I, or N, in the "Length" box, enter the field length.
6. If the data element is an array, in the "Dimensions" text box, select the number of dimensions (1-3).
   The "Lower Bounds" and "Upper Bounds" text boxes appear for each dimension.
7. Enter the lower bound and upper bound for each dimension in the array.
8. Choose **OK**.
   The "Define Local/Parameter Data" dialog box appears. The data element is displayed in the information box.

Modifying a Data Element

▶ **To modify a data element for a map**

1. From the **Map** menu, choose **Local Data** or **Parameter Data**.
   The "Define Local/Parameter Data" dialog box appears.
2. In the information box, select the data element to be modified.
3. Choose **Modify** to modify the data definition.
   The "Data Definition" dialog box appears.
4. Modify the data definition as required.
5. Choose **OK**.
   The "Define Local/Parameter Data" dialog box appears. The modified data element is displayed in the information box.

Removing a Data Element

To delete a data element for a map

1. From the **Map** menu, choose **Local Data** or **Parameter Data**.
   The "Define Local/Parameter Data" dialog box appears.
2. In the information box, select the data element to be deleted.
3. Choose **Remove** to delete the data definition.
   The data definition is removed from the information box.
4. Choose **OK**.

Testing Maps

Once you have created and successfully saved a map, you can test it so see how it will appear in an application.

To test a map

1. Open the map to be tested.
   The map editor is opened.
2. From the **Object** menu, choose **Test**.
   The Natural output window is opened and the map appears as it would in the application.
3. Double-click the output window or press **ENTER** or **ESCAPE** to return to the map editor.

Previewing Maps

Maps which are larger than the area shown in the map editor window can be scrolled using the scroll bars on the right, and at the bottom of the map editor window. Extremely large maps can be displayed in the map editor window by using the "Preview mode" function.

When preview mode is active, the display size of the map fields is reduced so that the entire map fits into the map editor window. All map editor functions work normally when preview mode is in effect.

To display a map in preview mode

- From the **Window** menu, choose **Preview Mode**.
  The map is displayed in preview mode.

Flipping Maps

Note:
This menu item is only applicable to right-to-left usage in the Natural International Version.

It can be used to flip/toggle the map editor’s editing/display mode.

Modifying the Map Profile

With the map profile, you set the profile parameters for the active map. These parameters are saved with the map.

To set profile parameters

1. From the **Map** menu, choose **Map Profile**.
   The "Map Editor Profile" dialog box appears.
2. Set profile parameters. Each parameter is described in the table below.
3. Choose **OK** to save the profile for the active map.

Map editor profile option settings are described in the following table.

Note:
The settings for "Zero printing" and "Upper case" are copied into the field definition when a new field is created. These settings can be modified for each new field.

Option	Setting	Explanation
Format:		
Page Size		The number of map lines to be edited (1 - 250). If "Std. Keys" is selected, the number of lines is restricted to the range 3 - 250. For a map that is output with a WRITE statement, specify the number of lines of the logical page output, not the map size. The map can then be output several times on one page.
Line Size		The number of map columns (5 - 249).
Column Shift		Column shift (0 or 1) to be applied to the map. This feature can be used to address all 80 columns on a 80-column screen (Column Shift = 1, Line Size = 80).
Layout		The name of the map that serves as standard layout for the current map. You can use this option to simplify the creation of many similar maps by creating one map as the basic layout map with a set of fields to be used by the other fields. In all similar maps, you specify the name of the standard layout map and you only add the fields that are specific to the new map. See also the "Dynamic Layout" option below.
Decimal Char	"."	The character to be used as the decimal notation character. The decimal notation character can only be changed with the GLOBALS command on the **Library** menu, or by entering the "GLOBALS=" command in the command line.
Print Mode		The default print mode for variables. This value is copied into the field definition when a new field is created.
	C	An alternative character set is to be used
	I	Inverse print direction
	N	Standard print direction
	Blank	No print mode
Standard Keys	**On**	The last two lines of the map remain empty so that function key specifications can be entered at execution time.
	Off	All lines are used for the map.
Upper Case	**On**	Input is converted to upper case during map execution.
	Off	Input is not converted to upper case during map execution.

Option	Setting	Explanation
Field Sensitive	**On**	The consistency check for a map field is made as soon as the field is filled by the user.
	Off	Field checking is performed when the map is filled completely.
Dynamic Layout	**On**	If you have specified the name of a standard layout map in the "Layout" field, you use this option to determine that fields are imported dynamically into the layout at compile time. This means that you can alter your standard layout and this change will automatically be reflected in all similar maps using this layout.
	Off	Do not use the standard layout dynamically.
Zero Printing	**On**	Print numeric fields that contain all zeroes (print one zero, right justified).
	Off	Suppress printing of numeric fields that contain all zeroes.
Right Justify	**On**	Numeric and alphanumeric fields taken from a user view or a data definition are right justified.
	Off	Numeric and alphanumeric fields taken from a user view or a data definition are not right justified.
Manual Skip	**On**	The cursor is not moved automatically to the next field in the map at execution time, even if the current field is completely filled.
	Off	The cursor is moved automatically to the next field in the map at execution time.
Context:		
Device Check		A device name can be viewed in this field.
WRITE statement	**On**	The result of the map definition process is a WRITE statement and the resulting (output) map can be invoked from a program using a WRITE USING FORM statement. Empty lines at the end of the map are automatically deleted so that the map can be output several times on one page.
	Off	The result of the map definition process is an INPUT statement and the resulting map can be invoked from a program using an INPUT statement.
Help Routine		The name of the helproutine that is called at execution time when the help function is invoked for this map (global help for the map).
Help Param		Enter the name of the "help" parameter that will be called at execution time when the help function is invoked. The "help" parameter can only be defined when the helproutine is specified. **Note:** If the combined length of the "Help Routine" and the "Help Param" exceeds 19 characters, it is truncated by the map editor.
as field default	**On**	The helproutine specified for the map applies as the default to each individual field on the map, that is, the name of each field is passed individually to the helproutine. This option can only be chosen if the helproutine is specified.
	Off	The name of the map is passed to the helproutine.
Help Text	**On**	The map is marked as help text.
	Off	The map is not marked as help text.
Position		
Line		The position (line, vertical) where the help map is being output.
Column		The position (column, horizontal) where the help map is being output.

Option	Setting	Explanation
AutoRuleRank		Define the default rank for any newly created processing rule in the map. (Later on, you can alter ranks individually).
Filler Characters:		
Optional, Partial		Indicates that a field is optional and can be partially filled.
Optional, Complete		Indicates that a field is optional but must be filled completely if it is used.
Required, Partial		Indicates that a field is required but can be partially filled.
Required, Complete		Indicates that a field is required and must be filled completely. **Note:** If you enter a field using the mouse, the cursor will be placed to the right of the filler character. The filler character is blank by default, and the field appears to be unmodifiable. Press **BACKSPACE** to delete the blank, and enter a new filler character.

Setting Editor Options

You can set preferences for various editor options. These settings are taken as default values each time you start the Map Editor.

To change options for the Map Editor

- From the **Tools** menu, select **Options**, and then in the option dialog select the tab "Map Editor".

Status Bar

Use this option to display the status bar at the top of the editor window. For information on the status line contents, see Status Bar Information.

Ignore Field Mode "Undef."

Use this option to have those fields ignored by a CHECK or STOW command, that have been created on the map but have not yet been named or defined; that is, they have only the values generated by the map editor.

Font

Use this option to select font types for maps. If you select a particular font for a map, this font is only output when you have defined the same font for the Natural output window. Otherwise the map is output with the default font.

Fixed fonts only

Use this option to specify that only fixed-width fonts are to be selectable in the "Font" dialog box (see above). When not selected, you can choose proportional-width fonts in addition to the fixed-width fonts.

Status Bar Information

The status bar appears at the top of the window where the map is edited. It displays the following information:

- **Name** - The name of the active field.
- **Row** - The top-most row in which the active field begins.
- **Col** - The left-most column in which the active field begins.
- **Len** - The length of the active field in characters.
- **Format** - The format of the active field.

Data Area Editor

The following topics are covered below:

- General Information
- The Column Header
- Generating Copycode from a Data Area
- Modifying Data Definitions
- Searching for Data Fields
- Setting Editor Options
- Navigating

See also:

- Data Area Editor Accelerators

General Information

A data area is a module containing the descriptions of data (data definitions) to be used by a Natural program, subprogram, subroutine, helproutine, dialog or class. It usually contains a declaration of user-defined variables and constants as well as referenced database fields in the form of data definition modules (DDMs).

Three different types of data areas can be used:

- **Local data areas** (LDAs) are used to define the data to be used within a single Natural program.
- **Global data areas** (GDAs) are used to define the data to be used by one or more Natural programs.
- **Parameter data areas** (PDAs) are used to specify the data parameters to be passed between a Natural program and a subprogram, external subroutine, helproutine, dialog or method.

All data areas are created and edited with the data area editor.

The data area editor window is used to enter and edit the field definitions that comprise a data area. The title bar at the top of the window includes the name of the data area (or "Untitled" if the data area has not been named), and the data area type; that is, local, global, or parameter. For example:

LDA01 [Local Data Area]

You can define fields and insert them into the new data area, and you can import fields from any cataloged DDM in any Natural library or from the Predict server into the new data area.

Column Header

The column header line contains the column headers for the data area fields. Notice that the column header for the field name changes, depending on the type of field that is selected.

Column Header	Description
I	Information field: R - Field is a "by value result" parameter. (Only allowed in parameter data areas.) V - Field is a "by value" parameter. (Only allowed in parameter data areas.) X - More information is available for the field. An edit mask, header, or init value is defined for the field.
T	Type of field: B - Block C - Constant or Counter variable in views G - Group Field H - Handle of dialog element or object M - Multiple value field O - Handle of object P - Periodic group S - Structure U - Global unique ID (GUID) V - View blank - Elementary field * - Comment
L	Level of the field (1 - 9).
Name of data field	Name of the view, group, periodic group, multiple-value field, constant, count variable, handle, block or elementary field.
F	Format of the field.
Len	Length of the field.
Index/Comment	Array indices and field comment.
Name of DDM	DDM name.
Parent/Comment	Parent of a block and field comment.

Generating Copycode from a Data Area

This function creates a copycode object containing a DEFINE DATA statement which corresponds to the current data area. You can then edit the generated copycode with the program editor.

To generate a copycode

1. Open the data area from which a copycode is to be generated.
2. From the **Object** menu, choose **Generate**.
   The copycode is generated and given the name "Untitled".

Modifying Data Definitions

- Modifying Fields
- Inserting Fields

Modifying Fields

Selecting Fields

To select a field in the data area editor

- Click on the field.

To select a range of fields

1. Point to the first field to be selected.
2. Drag the cursor to the last field you want to select.
3. Release the mouse button.

To select all fields in the data area editor

- From the **Edit** menu, choose **Select all**.

Copying Fields

Fields can be copied and pasted within the same data area or another data area.

To copy fields in the data area editor

1. Select the field(s) to be copied (see Selecting Fields).
2. From the **Edit** menu, choose **Copy**.
   Or click the **Copy** toolbar button.
   Or press **CTRL+C**.
   The field is copied to the clipboard and can now be pasted within the same data area or another data area. For instructions on pasting text, see Pasting Fields.

To copy data area field(s) to a Natural object handled by the program editor

1. From the **Objects** menu, choose **List**.
2. Select the lines(s) to be copied.
3. From the **Edit** menu, choose **Copy**.
   Or click the **Copy** toolbar button.
   Or press **CTRL+C**.
   The field is copied to the clipboard and can now be pasted to a program editor object.

Cutting Fields

The cut function can be used to delete fields from a data area or to move fields within a data area. When a field is cut, it is taken from the object and placed on the clipboard. It remains there until the next cut or copy operation is performed, at which time it is irretrievably discarded from the clipboard to make way for the next cut/copied field.

To cut fields from the data area editor

1. Select the field(s) to be cut (see Selecting Fields).
2. From the **Edit** menu, choose **Cut**.
   Or click the **Cut** toolbar button.
   Or press **CTRL+X**.
   The field is cut to the clipboard and can now be pasted within the object or to another object. For instructions on pasting text, see Pasting Fields.

Pasting Fields

To paste or cut fields as described above

1. Select the field after which or before which the field is to be pasted.
   Whether the field is pasted before or after the selected field is determined by the **Insert** option specified in the **Options, Data Area Editor** menu. For more information on the insert options, see Setting Editor Options.
2. From the **Edit** menu, choose **Paste**.
   Or click the **Paste** toolbar button.
   Or press **CTRL+V**.
   The selected fields are pasted after/before the selected field.

Deleting Fields

To delete fields in the data area editor

1. Select the field(s) to be deleted (see Selecting Fields).
2. From the **Edit** menu, choose **Delete**.
   Or click the **Delete** toolbar button.
   Or press **DEL**.
   If delete messages are active, you are requested to confirm deletion. Otherwise, the fields are deleted without confirmation.

Inserting Fields

There are several ways to insert fields into the data area editor.

To insert fields

1. Select a field.
   Inserted fields are placed before or after the selected field, depending on whether **Insert before** or **Insert after** is selected in the **Options, Data Area Editor** menu (see Setting Editor Options).
2. From the **Insert** menu, choose a field type.
   Or press **INS** and, in the "Insert Field" dialog box that appears, select a field type, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert Field" dialog box that appears, select a field type, and then choose **Add**.
3. You can insert the field repeatedly by choosing **Add**.
4. Use **Cancel/Quit** to quit the dialog box.

The remainder of this section describes the definition of each type of field.

Inserting a Data Field

To insert a data field into the active data area

1. From the **Insert** menu, choose **Data Field**.
   Or press **INS** and, in the "Insert Field" dialog box that appears, select **Define a Data Field**, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert an Entry" dialog box that appears, select **Data Field**, and then choose **OK**.
   The "Data Field Definition" dialog box appears.
2. In the "Level" text box, enter a level number (1-9). The level number cannot be more than one level higher than the previous level.
3. In the "Field" text box, enter a name for the field. The field name must be a valid Natural identifier. (See section Object Naming Conventions).
4. In the "Format" drop-down list box, select the desired format for the data field. For a list of valid formats, see Definition of Format and Length in the Natural Reference documentation.
5. If the "Dynamic" check box is activated, the field length is set dynamically. This is only available for alphanumeric and binary fields. In this case the length text box will be deactivated.
6. In the "Length" text box, enter the field length. The formats C, L, D and T do not require a length definition. If you enter an invalid length, a message displays valid lengths for the specified format.
7. To define an array, choose "Array" and see Defining an Array.
8. To specify an edit mask, in the "Edit Mask" text box, enter its name. Edit masks are optional.
   If an edit mask is used, it must conform with Natural syntax rules and be valid for the field length and format. See Edit Masks in the Natural Reference documentation.
9. To specify a header for the data field, in the "Header" text box, enter the name of the header.
10. To document the data field, in the "Comment" text box, you can enter a comment. The comment can be up to 29 characters long, minus the length of the array definition, if any.
11. To initialize a value for the data field, choose **Initialize** and see Initializing the Value for a Data Field. This definition does not apply to parameter data areas.
12. Choose **Add**.

Defining an Array

The "Array Definition" dialog box displays the specified field name and type.

To define an array

1. In the "Dimensions" drop-down list box, select the number of dimensions for the array (1, 2, or 3).
   To delete an array definition, select 0.
2. In the "Lower Bound" text box, enter the lower bound for each dimension.
   You can enter a numeric constant with a positive or negative offset (as lower bound), which must be defined before the edited field. You can also enter a variable name if the edited field is a multiple field or a periodic group.
   A constant or variable name is only allowed if no constant or variable is specified in the "Occurrences" text box.
3. In the "Occurrences" text box, enter the number of occurrences for each dimension. You can enter a numeric constant with a positive or negative offset.
   A constant or variable name is only allowed if no constant or variable is specified in the "Lower Bound" text box.
4. Choose **OK**.
   The "Array Definition" dialog box closes.

Initializing the Value for a Data Field

In the "Initialize" dialog box, you can enter the value(s) for a data field in two different ways: single-value and free-form mode.

In single-value mode, you enter the values in a structured way. In free-form mode, you enter the values just as you would in a DEFINE DATA statement.

To initialize the value for a data field in single-value mode

1. Select the **Single Value** option button.
2. If the data field is an array, then an "Index" drop-down box is displayed for each dimension. If not, then go to Step 4.
3. Select the index element to which you want to assign a value by selecting a value from each index box.
   For example, to select array element (3,2) located in the third row and second column of a two dimensional array, select 3 in the first index box and 2 in the second index box.
4. In the "Value" text box, enter the value.
   Make sure that the value corresponds to the field type. Note that parentheses, apostrophes or value prefixes (for example, H for Hex, D for Date, or T for Time) are not required.
5. If the data field is an array, choose **Accept** to apply the value and increment the index by one. Repeat Steps 3 and 4 until you have initialized all of the array fields you want. Then choose **OK**.
   If the data field is not an array, choose **OK**.
   The "Initialize" dialog box closes.

Initializing Values in Free-form Mode

To initialize the value for a data field in free-form mode

1. Select **Free Form Entry**.
   An edit box is displayed.
2. Enter the values in INIT syntax just as you would in a DEFINE DATA statement .

Inserting a Global Unique ID

This function is only possible with LDAs and GDAs.

To insert a global unique ID

1. From the **Insert** menu, choose **Global Unique ID**.
   Or press **INS** and, in the "Insert Field" dialog box that appears, select **Global Unique ID**, and then choose **OK**.
   Or click the **Inserts a Field** toolbar button, in the "Entry Definition" dialog box that appears, select **Global Unique ID**, and then choose **OK**.
   The "Global Unique ID Definition" dialog box appears.
2. In the "Level" text box, enter a level number (1-9). The level number can only be one level higher than the previous level.
3. In the "Name" text box, enter a name for the global unique ID. The name must be a valid Natural identifier. (See section Object Naming Conventions.)
4. In the "Comment" text box, you can enter a comment.
5. Choose **ADD**.
   The global unique ID is generated as a Natural constant with length A36.

Inserting a Constant

Note:
You cannot insert constants into parameter data areas.

To insert a constant into the active data area

1. From the **Insert** menu, choose **Constant**.
   Or press **INS** and, in the "Insert Field" dialog box that appears, select **Define a Constant**, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert Field" dialog box that appears, select **Constant**, and then choose **OK**.
   The "Constant Definition" dialog box appears.
2. In the "Name" text box, enter a name for the constant. The constant name must be a valid Natural identifier. (See section Object Naming Conventions).
3. In the "Format" drop-down list box, select the desired format for the constant. For a list of valid formats, see Definition of Format and Length in the Natural Reference documentation.
4. In the "Length" text box, enter the field length. The formats C, L, D and T do not require a length definition. If you enter an invalid length, a message displays valid lengths for the specified format.
5. To define an array, choose **Array** and see Defining an Array.
6. To specify an edit mask, in the "Edit Mask" text box, enter its name. Edit masks are optional. If an edit mask is used, it must conform with Natural syntax rules and be valid for the field length and format. Edit Masks are described in the Natural Reference documentation.
7. To specify a header for the constant, in the "Header" text box, enter the name of the header.
8. To document the constant, in the "Comment" text box, you can enter a comment. The comment can be up to 32 characters long, minus the length of the array definition, if any.
9. Choose **Initialize** and see Initializing the Value for a Data Field.
10. Choose **Add**.

Inserting a Data Block

A data block is a collection of variables and/or DDMs. Blocks can be defined only for global data areas.

To insert a data block into the active global data area

1. From the **Insert** menu, choose **Block**.
   Or press INS and, in the "Insert Field" dialog box that appears, select **Block**, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert Field" dialog box that appears, select **Define a Block**, and then choose **OK**.
   The "Define a Block" dialog box appears.
2. In the "Name" text box, enter a name for the data block.
3. In the "Parent" text box, enter the name of the parent block.
   Note:
   If you use a parent block, it must be defined in the current data area. Otherwise, a syntax error occurs.
4. In the "Comment" text box, enter a comment concerning the block. The comment can be up to 32 characters long, minus the length of the parent block name.
5. Choose **OK** to save the completed block definition.
   The "Define a Data Field" dialog box appears.
6. Define the data fields belonging to the block.
   For more information on defining data fields, see Inserting a Data Field.

Inserting a Data Structure

A data structure, also known as a group, consists of data fields and other structures. The maximum number of levels in a data structure is 9.

To insert a data structure into the active data area

1. From the **Insert** menu, choose Structure.
   Or press **INS** and, in the "Entry Definition" dialog box that appears, select **Define a Structure**, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert Field" dialog box that appears, select **Structure**, and then choose **OK**.
   The "Structure Definition" dialog box appears.
2. In the "Level" text box, enter a level number (1-9). The level number cannot be more than one level higher than the previous level.
3. In the "Name" text box, enter a name for the structure. The structure name must be a valid Natural identifier. (See section Object Naming Conventions).
4. To define an array, choose **Define Array** and see Defining an Array.
5. To document the structure, in the "Comment" text box, you can enter a comment. The comment can be up to 32 characters long, minus the length of the array definition, if any.
6. Choose **OK** to save the completed data structure definition.
   The "Data Field Definition" dialog box appears.
7. Define the data fields belonging to the structure.
   The level number for a data field cannot be more than one level higher than the previous level.
   For more information on defining data fields, see Inserting a Data Field.

Inserting a Handle

For a handle, you can define the type "Dialog Element" or "Object".

To insert a handle into the active data area

1. From the **Insert** menu, choose **Handle**.
   Or press **INS** and, in the "Insert Field" dialog box that appears, select **Define a Handle**, and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Entry Definition" dialog box that appears, select **Define a Handle**, and then choose **OK**.
   The "Handle Definition" dialog box appears.
2. In the "Level" text box, enter a level number (1-9). The level number cannot be more than one level higher than the previous level.
3. In the "Name" text box, enter a name for the handle. The handle name must be a valid Natural identifier. (See section Object Naming Conventions).
4. To define an array, choose **Define Array** and see Defining an Array.
5. To define a handle of the type dialog element, in the "Type" field, select the **Dialog Element** toggle button and open a list box. There, choose a dialog element.
6. To define an object handle, in the "Type" field, select the **Object** toggle button.
7. To document the handle, in the "Comment" text box, you can enter a comment. The comment can be up to 32 characters long, minus the length of the array definition, if any.
8. Choose **ADD** to save the completed handle definition.
   The handle is inserted into the data area and an empty "Define Handle" dialog box appears.
9. Define another handle or choose **Cancel** to exit.
   For further information on object handles, see the NaturalX documentation.

Inserting a Comment

Comments are used to document Natural source code and are ignored during processing.

You can insert a comment into the active data area, before or after a selected field. The "Define Comment" dialog box allows you to define the comment that you want to insert.

To insert a comment into the active data area

1. From the **Insert** menu, choose **Comment**.
   Or press INS and, in the "Insert Field" dialog box that appears, select "Comment", and then choose **OK**.
   Or click the **Insert a Field** toolbar button, in the "Insert Field" dialog box that appears, select **Comment**, and then choose **OK**.
   The "Comment Line Definition" dialog box appears.
2. In the "Comment" text box, enter text up to 69 characters long.
3. Choose **Add** to insert the completed comment.
   The comment is inserted into the data area and an empty "Comment Line Definition" dialog box appears.
4. Define another comment or choose **Cancel/Quit** to exit.

Modifying a Field Definition

1. Double-click the field name.
   Or select the field name and, from the "Field" menu, choose "Modify".
   A dialog box appears. The type of dialog box depends on the type of field.
2. Modify the field attributes as necessary and then choose **OK** to save your changes.
   For further information on defining data fields, see Inserting a Data Field.

Defining a Counter Variable

A counter variable is used to specify how many occurrences exist for a multiple field or periodic field.

To create a counter variable

1. Select the a multiple field or periodic field.
2. From the "Field" menu, choose "Counter".
   A counter field appears in the data area. It is labeled "C".

Redefining a Field Definition

Redefining a field enables you to convert the format of a field or divide a single field into segments.

To redefine a field definition from one type to another

1. Select the field name and, from the **Field** menu, choose **Redefine**.
   The "Insert Redefine" dialog box appears.
2. To define a structure, select **Structure**.
   To define a data field, select **Data Field**.
   To define a comment, select **Comment**.
3. Choose **OK**.
   A corresponding dialog box appears.
4. Enter the input required and choose **OK**.
   The "Insert Redefine" dialog box appears again.
5. Repeat Steps 2 through 4 until no more bytes are available or until the redefinition is complete.

Note:
To specify filler bytes, in the "Name" text box of the "Data Field Definition" dialog box, enter nX.

Importing Data Fields into a Data Area

Data fields can be imported from any Natural object or DDM in any library or from a Predict server.

To import selected data fields from a DDM into the active data area

1. From the **Insert** menu, choose **Import**.
2. The "Import" dialog box appears. The name of the current library is displayed in the "Library" list box.
3. If the object containing the fields you want to import is located in a different library, open the "Library" list box and select the library.
4. From the "Type" group frame, select the type of Natural object from which you want to import fields.
   A list of all Natural objects in the current library of the type you selected appears in the "Object List" box.
5. Select the object that contains the fields you want to import. The fields in the selected object appear in the "Data Fields" list box.
6. Select the fields that you want to import.
7. Choose **Import**.
8. Choose **Quit**.
   The dialog box closes and the fields appear in the upper left corner of the map editor window. You can move the fields around within the map.

To import selected data fields from a DDM into the active data area

1. Indicate where the imported fields should be placed by selecting a field in the data area. Imported fields are placed before or after the currently selected field, depending on whether **Insert before** or **Insert after** is selected in the **Options** menu.
2. From the **Insert** menu, choose **Import**. Or click the **Import Data Field** toolbar button.
   The "Import Data Field" dialog box appears.
3. In the "Library" drop-down list box, select the library containing the DDM to be imported.
   A list of all the cataloged DDMs in the selected library appears in the "DDM list" list box.
4. From the "Type" group frame, select "View". A list of all Natural DDMs in the current library appears in the "Object List" box.
5. In the "Object List" list box, select the DDM that contains the fields you want to import. A list of all data fields in the selected DDM appears in the "Data Fields" list box.
6. From the "Data Fields" list box, select the fields you want to import into the data area window.
7. Choose **OK**.
   The "View Definition" dialog box appears.
8. In the "Name of View" text box, enter the name to be used for the view in the data area.
9. In the "Comment" text box, enter the comment to accompany the view in the data area.
10. Choose **OK**.
    The dialog box closes and the fields are imported into the data area.

Importing Periodic Groups

If you are importing a periodic group, or a multiple-value field, the "Define Occurrences" dialog box appears when you choose **OK**. You can change the lower bound and the number of occurrences. Then choose **OK**.

Searching for Data Fields

In large data areas, it is often difficult to locate data fields. Using the search function you can flexibly search for data field names. If it should be necessary to replace a frequently occurring field name with another, you can use the combined search and replace function.

1. From the **Edit** menu, choose **Find**.
   Or click the "Find the specified text" toolbar button.
   Or press **CTRL+F**.
   The "Find" dialog box appears.
2. In the "Find Field Name" text box, enter the string to be searched for.
3. If you want the search string to be found as a whole word only and not as part of other words, select the "Match Whole Words Only" text box.
   If this box is left unselected, all occurrences of the string will be found.
4. In the "Direction" group frame, click the search direction up or down to specify whether the search will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
5. Choose **Find Next**.
   If no instance of the text searched for is found, a corresponding message is displayed.
   If an instance of the search string is found, it will be displayed.

To search for additional instances of the search string in the object

- From the **Edit** menu, choose **Find Next**.
  Or press **F3**.

Searching for and Replacing Data Field Names

To search for and replace a text string in the active data area window

1. From the **Edit** menu, choose **Replace**.
   Or click the **Replace Text** toolbar button.
   Or press **CTRL+H**.
   The "Replace Data Field" dialog box appears.
2. In the "Find Field Name" text box, enter the string to be searched for.
3. In the "Replace with" text box, enter the replacement string.
4. If the search string is to be found as a whole word only and not as part of other words, select the "Match whole words only" text box.
   If this box is left unselected, all occurrences of the string will be found.
5. In the "Direction" group frame, choose the search direction up or down to specify whether the search and replace will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
6. Choose **Replace**.
7. Choose **Close** to exit the dialog.
   If no instance of the text searched for is found, a corresponding message is displayed.

Repeat Replace

- To do so, from the **Edit** menu, choose **Replace Next**.
  Or press **CTRL+F3**.

Replace All

With the "Replace All" function one can replace all strings at one time.

Note:
If "Replace All" is executed, all the found strings will be replaced. The "UNDO" function is not available.

Setting Editor Options

You can set preferences for various editor options. These settings are taken as default values each time you start the Data Area Editor.

To set options

- From the **Tools** menu, select **Options** and then in the options dialog select the tab "Data Area Editor".

Insert before

Insert a field before the currently selected field.

Insert after

Insert a field after the currently selected field.

Status bar

Display/hide the status line at the top of the data area editor.

The status line displays (1) the number of bytes used by the data area in the source area, (2) the location of the selected line, and (3) the total number of lines in the data area.

Note:
The data area editor can handle a maximum of 9999 lines. If this number is exceeded, an error occurs.

Column header

Display/hide the column header at the top of the data area definition.

Navigating

To navigate in the level order within the data area select the **Next Level** or **Previous Level** in the **View** menu, or use **CTRL+SHIFT+I** or **CTRL+SHIFT+J**.

DDM Editor

The following topics are covered below:

- The DDMs Window
- Adding DDMs
- Modifying DDM Contents
- Searching for DDM Fields
- Modifying DDM Fields
- Editor Window Layout
- Setting Editor Options

DDMs Window

The "DDMs" node within the logical view displays a list of all DDMs.

To read the values of a DDM

1. In the "left window" choose "System Libraries" and open the folder "DDMs" in the "SYSEXDDM" library.
2. With a double-click open the desired file.

The following information is displayed for each DDM:

- **T** - Type.
- **L** - Level state.
- **Name** - Field name.
- **F** - Field format.
- **Len** - Field length.
- **S** - Suppresion.
- **D** - Descriptor type.

A DDM (data definition module) is a set of field definitions for a database file. A DDM can be created from a database file or from other DDMs.

DDMs are used to describe any type of database file, and are not restricted to Adabas database files. Some options described in this section only pertain to Adabas, and can be ignored if a different database system is being used.

Adding DDMs

- Adding DDMs from Adabas Databases
- Adding DDMs from SQL Databases
- Additional Options for VSAM Files

Adding DDMs from Adabas Databases

▶ **To create a new DDM for an Adabas database**

1. From the **Object** menu, choose **New** and then, from the cascading menu, choose **DDM**.
   Or click the **Create a new DDM** toolbar button.
   The "New DDM - Select Database" dialog box appears.
2. If the "DBID - Database Type " list box is active, you can choose a DBID from the list. You can also enter a DBID in the range of 0 to 65535
   (except 255) in the "DBID edit control" field. Click **OK** to acknowledge.When you enter "DBID", which does not exist in the listbox, you can also select a type from the combobox. The "New Adabas DDM" dialog box appears in which the file number can be entered.
3. In the "File number" field enter a file number in the range of 1 to 5000 and choose **OK**.
   The DDM editor appears. If the database ID and file number correspond to an existing database file (FDT), then the fields contained in that database file are displayed. The DDM can then be edited as required.

Adding DDMs from SQL Databases

▶ **To create a new DDM for an SQL database**

1. From the **Object** menu, choose **New** and then, from the cascading menu, choose **DDM**.
   Or click the **Create a new DDM** toolbar button.
   The "New DDM - Select Database" dialog box appears.
2. From the "DBID - Database Type" list box, choose an ID and choose **OK**.
   A "New SQL DDM" dialog box appears. It contains the text boxes "Table Owner" and "Table Name" and is used to select the SQL table to be added.
3. To list selected SQL tables, enter a pattern using the wildcard symbol ("*").
   To list all SQL tables for selection, in the text boxes, leave the asterisks and choose **OK**.
   A list box displays all SQL tables for selection according to the selection criteria specified above.
4. In the list box, select the desired SQL table and choose **OK**.
5. If you are accessing this SQL database for the first time in this session, then a "Database Logon" window appears (dependant from the SQL database). Enter the user ID and password for the database and choose **OK**.
   The SQL table is read into the DDM editor. It can be edited as required.

A name is generated automatically for the SQL DDM. It is a combination of the table owner and the table name and cannot be altered.

Note:
With the Natural program DDMGEN you can generate serveral DDMs in a particular library without using the editor.(available only under a local environment)

Note:
When working under a remote environment, this function is available only if Natural for DB2 or Natural for SQL/DS is installed. It is used to generate DDMs from DB2 or SQL/DS tables and is described in the documentation Natural for DB2 and Natural for SQL/DS respectively.

Additional Options for VSAM Files

The additional options for VSAM files consist of two parts: VSAM File Information and VSAM File Organization.

To define the VSAM file information

1. In the "VSAM file name" text box, enter the DDNAME/FCT entry as defined to the TP monitor.
2. If the "VSAM view" check box is set, this DDM represents a logical DDM. If it is unchecked, it represents a physical DDM.
   The following applies only if the "VSAM view" check box is set:
3. In the "Logical related to FNR" edit control, you can enter the file number of the physical DDM from which the logical file or DDM is derived.
4. In the "User defined prefix" edit control, you can enter the prefix value, which is to be assigned to the logical file.

To define the VSAM file organization

1. Set the type of the VSAM file by selecting one of the radio buttons.
2. If the "Compress file" check box is set, the file is to be compressed.
3. With the "Zones" combobox, you can select the zone for the VSAM file. ’F’ indicates, that all packed data are written to the VSAM file with the zone X’0F’. ’C’ indicates, that all packed values are written to the VSAM file with the zone X’0C’.

Note:
For more information concerning DDMs for VSAM, please refer to the Natural for VSAM documentation.

Data Conversion

Note: The following section applies under a local environment only!
If large and dynamic variables and/or fields are needed, please read the section DDM Generation and Editing for Varying Length Columns in the section Large and Dynamic Variables/Fields in the Natural Programming Reference documentation.

When a Natural program accesses data in a relational database, RDBMS-specific data types are converted to Natural data formats, and vice versa. The tables in this section show how Natural data formats correspond to data types in the following RDBMS’s:

- Adabas D
- Adabas SQL Server
- DB2
- INFORMIX
- INGRES
- ORACLE
- SYBASE and Microsoft SQL Server

Adabas D

RDBMS Data Type	**Natural Format/Length**
boolean	L
char (*n*)	A*n*
date	A10
fixed (*p*,*q*)	N*p-q*,*q*
float	F8
integer	I4
long	A, DYNAMIC
long varchar	A, DYNAMIC
smallint	I2
string	A*n*
time	A8
timestamp	A26
varchar	A*n*

Adabas SQL Server

RDBMS Data Type	Natural Format/Length
char(5)	A5
char(253)	A253
decimal(5)	N5
decimal(10.4)	N(6.4)
double precision	N(10.6)
float(1...21)	N(2.6)
float(22...53)	N(10.6)
integer	I4
numeric(5)	N5
numeric(10.4)	N(6.4)
real	N(2.6)
smallint	I2

DB2

RDBMS Data Type	Natural Format/Length
date	A10
decimal(5)	N5
decimal(10.4)	N(6.4)
fixed character(5)	A5
float	F*n*
graphic	2*A*n*
longvar	A, DYNAMIC
longvarg	A, DYNAMIC
large integer	I4
scientific notation	N(10.6)
small integer	I2
special data	A253
system date and time	A10
time	A8
timestmp	A26
varchar	A*n*
varg	2*A*n*

INFORMIX

RDBMS Data Type	**Natural Format/Length**
byte	B*n*
char(*n*)	A*n*
date	A10
datetime	A26
decimal(*p*,*q*)	N*p-q*,*q*
double precision	F8
float	F8
integer	I4
interval	A17
money	N(14.2)
numeric	N*p-q*,*q*
real	F4
serial	I4
smallint	I2
smallfloat	F4
text	A*n*
varchar(*n*)	A*n*

INGRES

RDBMS Data Type	Natural Format/Length
byte varying	B*n*
c(*n*)	A*n*
char (*n*)	A*n*
date	A10
double precision	F8
float	F8
float4	F4
integer	I4
integer1	I1
long byte	B, DYNAMIC
long varchar	A, DYNAMIC
money	N(12.2)
object_key	B16
real	F4
smallint	I2
table_key	B8
text (*n*)	A*n*
varchar (*n*)	A*n*

ORACLE

RDBMS Data Type	**Natural Format/Length**
char (*n*)	A*n*
date	A10
decimal (*p*,*q*)	N*p*-*q*,*q*
double precision	F8
float	F4
integer	I4
long	A, DYNAMIC
long raw	B, DYNAMIC
number	N*n*
nvarchar2	A*n*
raw (*n*)	B*n*
real	F4
rowid	A*n*
smallint	I2
varchar	A*n*
varchar2 (*n*)	A*n*

SYBASE and Microsoft SQL Server

RDBMS Data Type	Natural Format/Length
binary (*n*)	B*n*
bit	N1
char (*n*)	A*n*
datetime	A26
float	F8
image	B*n*
int	I4
money	N(15.4)
nchar (*n*)	A*n*
nvarchar (*n*)	A*n*
real	F4
smalldatetime	A26
smallint	I2
smallmoney	N(6.4)
text	A*n*
timestamp	B8
tinyint	I2
varbinary (*n*)	B*n*
varchar (*n*)	A*n*

Modifying DDM Contents

Note:
The access to DDMs may be restricted when Natural Security has been installed. Within the DDM security profile, there may be a definition of whether a DDM may be modified only by specific users (DDM modifiers) or the owners of the security profile.

For further information refer to the Natural Security documentation for OpenVMS, UNIX and Windows NT, under "DDM Restrictions" section DDM Security Profiles.

- Selecting Fields
- Selecting Attributes in Fields
- Copying Fields
- Cutting Fields
- Pasting Fields
- Deleting Fields
- Inserting Fields

Selecting Fields

To select one or more fields in the DDM editor

1. Put the mouse pointer on the left margin of the field and click.
   Click on the first field to be selected and press **SHIFT+DOWN ARROW** or **SHIFT+UP ARROW**.
   The field is highlighted.
2. To select more contiguous fields, press **UP ARROW** or **DOWN ARROW**.
   The contiguous fields are highlighted.

To select all of the fields in the DDM editor

- From the **Edit** menu, choose **Select all**.
  All of the fields in the editor are highlighted.

Selecting Attributes in Fields

To select an attribute in a field in the DDM editor

- Put the mouse pointer on the attribute and click.
  Or use **TAB**, **SHIFT+TAB**, **UP ARROW**, and **DOWN ARROW** to navigate to the desired field.
  The field attribute is highlighted and surrounded by a box.

Copying Fields

To copy fields in the DDM editor

1. Select the fields you want to copy using the instructions found in Selecting Fields.
2. From the **Edit** menu, choose **Copy**.
   Or click the **Copy** toolbar button.
   Or press **CTRL+C**.
   The fields are copied to the clipboard and can be pasted within the same DDM or another DDM. For instructions on pasting fields, see Pasting Fields.

Cutting Fields

The cut function can be used to delete fields from a DDM or to move fields within/between DDMs. When text is cut, it is taken from the DDM and placed on the clipboard. It remains there until the next cut or copy operation is performed, at which time it is irretrievably discarded from the clipboard to make way for the next cut/copied field.

To cut fields in the DDM editor

1. Select the fields you want to cut using the instructions found in Selecting Fields.
2. From the **Edit** menu, choose **Cut**.
   Or click the **Cut** toolbar button.
   Or press **CTRL+X**.
   The fields are cut to the clipboard and can be pasted within the same DDM or another DDM. For instructions on pasting fields, see Pasting Fields.

Pasting Fields

The paste function is used to place a field at a specific position within a DDM after it has been copied or cut to the clipboard from another position within the same DDM or another DDM. A field which has been copied or cut to the clipboard can be pasted repeatedly without recopying it.

To paste the fields in the DDM editor

1. Cut or copy DDM fields as described in Cutting Fields or Copying Fields.
2. If the fields are to be pasted in another DDM, open the DDM.
3. Select the field after/before which the copied/cut fields are to be pasted.
   Whether the field is pasted before or after the selected field is determined by the "Insert" option specified in the "Options" menu. For more information on the insert options, see Setting Editor Options.
4. From the **Edit** menu, choose **Paste**.
   Or click the **Paste** toolbar button or press **CTRL+V**.
   The field is pasted in the DDM.
5. To paste the same field again, repeat Steps 2 through 4.

Deleting Fields

When a field is deleted, it is cut from the DDM but is *not* placed on the clipboard. There is no way to recover the field once deleted.

To delete fields from the DDM editor

1. Select the fields you want to delete using the instructions found in Selecting Fields.
2. From the **Edit** menu, choose **Delete**.
   Or click the **Delete** toolbar button.
   Or press **DEL**.
   The fields are deleted from the DDM and cannot be recovered.

Inserting Fields

You add fields to a DDM by inserting them. The insert function has two different modes depending on the context in which you use it.

- If you are editing a DDM which belongs to an active database, then you must select a field from the existing DDM for insertion. You can then modify this field.
- If you are editing a DDM which does not belong to an active database, then you can insert an empty DDM field which you fill as required.

In both cases, the field is inserted either before the selected field or after the selected field, depending on the current editor option setting.

To insert a field into the active DDM

1. In the DDM, select a field.
2. From the **Field** menu, choose **Insert**.
3. If you are editing a DDM which does not belong to an active database, then a blank line is inserted before/after the selected field.
   Enter a value for each field attribute.
   Use **TAB** to move from one column to the next.
   Whether the field is pasted before or after the selected field is determined by the **Insert** option specified in the **Options** menu. For more information on the Insert option, see Setting Editor Options.
4. If you are editing a DDM which belongs to an active database, then a "Field Selection List" window appears and one of the fields can be selected. The field selected in the field selection list is inserted before/after the field selected in the DDM.

Modifying DDM Fields

- Modifying Extended Attributes under a Local Enviroment
- Modifying Extended Attributes under a Remote Enviroment
- Displaying Descriptor Information
- Modifying a DDM Header
- Modifying Coupling Information

Modifying Extended Attributes under a Local Enviroment

To display and edit the extended attributes of the fields contained in the DDM

1. In the active "DDM" window, select a field.
2. From the **Field** menu, choose **Extended fields**.
   Or click the **Extended Fields** toolbar button.
   The "Extended Attributes" dialog box appears with the name of the selected field.
3. If a header exists for the selected field, it appears in the "Header" text box. You can edit the header, or add a header if none exists.
4. If an edit mask exists for the selected field, it appears in the "Edit Mask" text box. You can modify the edit mask, or add an edit mask if none exists.
   The edit mask must conform with Natural syntax rules and be valid for the field length and format. (See the section Edit Masks in the Natural Reference documentation.)
5. If a remark exists for the selected field, it appears in the "Remarks" text box. You can edit the remark, or add a remark if none exists.
6. To save any changes you have made to the current field choose **Save**.
7. To view and edit extended attributes for the next field in the DDM, choose **Next**.
8. To view and edit extended attributes for the previous field in the DDM, choose **Prev**.
9. To save and validate all field modifications and return to the "DDM" editor window, choose **OK**.

Modifying Extended Attributes under a Remote Enviroment

Note:
Applies only for DDM’s generated for VSAM

To display and edit the extended attributes of the fields contained in the DDM

1. In the active "DDM" window, select a field.
2. From the **Field** menu, choose **Extended fields**.
   Or click the **Extended Fields** toolbar button.
   The "Extended Attributes" dialog box appears with the name of the selected field.
3. If a header exists for the selected field, it appears in the "Header" text box. You can edit the header, or add a header if none exists.
4. If an edit mask exists for the selected field, it appears in the "Edit Mask" text box. You can modify the edit mask, or add an edit mask if none exists.
   The edit mask must conform with Natural syntax rules and be valid for the field length and format. (See the section Edit Masks in the Natural Reference documentation.)
5. If an alternate descriptor (Type A) or superdescriptor (Type X) is defined for the field, you can enter an alternative index name.
6. If the field is a multiple or periodic group field, you can specify the number of occurences in the "Maximum Occurence" edit control box.
7. If an alternate descriptor (Type A) or superdescriptor (Type X) is defined for the field, you can set the flags "Upgrade", "Unique Key", "Sort" and "Null" in the "Extended Attributes" dialog box.
8. If the field has a primary or secondary key descriptor (Type A) or superdescriptor (Type X), you can select the database shortname from the combo box.

9. If a remark exists for the selected field, it appears in the "Remarks" text box. You can edit the remark, or add a remark if none exists.
10. To save any changes you have made to the current field choose **Save**.
11. To view and edit extended attributes for the next field in the DDM, choose **Next**.
12. To view and edit extended attributes for the previous field in the DDM, choose **Prev**.
13. To save and validate all field modifications and return to the "DDM" editor window, choose **OK**.

Displaying Descriptor Information

With this function you can display the makeup of a subdescriptor field or a superdescriptor field.

To do so

1. In the descriptor field, select a descriptor type.
2. From the **Field** menu, choose **Sub-/Superdescriptor Info**.
   Or click the **Descriptor Definition** toolbar button.
   The "Descriptor Definition" dialog box appears with the name of the selected subdescriptor or superdescriptor next to "Field name".
3. Choose **OK** to exit the field definition dialog.

Modifying a DDM Header

To edit the DDM header information

1. From the **DDM** menu, choose **DDM header**.
   Or click the **DDM Header** toolbar button.
   The "DDM Header Information" dialog box appears with the names of the current DDM (Remote and Local)
   and library (Local only) appearing at the top of the box.
2. In the "DBID" drop down list box you can choose a DBID out of the list. You can also enter a DBID in the range of 0 to 65535 (except 255) in the "DBID edit control" field. When you enter "DBID", which does not exist in the listbox, you can also select a type from the "Type" combobox.
3. To change the file number to which the DDM is assigned, enter a new value in the range of 1 to 5000 in the "File ID" edit control field.
4. In the "Default Sequence" text box, enter a short name as default sequence.
   The system validates your entry based on the selected file number.
   Note for Adabas only:
   If the database is accessible, then the short name is verified to determine if it exists. If it does not exist, then a list is presented with a selection of valid short names. If the database is not active, a list cannot be generated.
5. Choose **OK** to save the new values.

Modifying Coupling Information

This option only applies for Adabas DDMs and has only informational character.

If you select this option, all files physically coupled to the displayed DDM are listed together with the short names of the descriptors used for coupling.

To define a coupled file

1. In the "File Name" text box enter the name of the coupled file.
2. In the "File Number" text box enter the number of the coupled file.
3. In the "From" edit control, enter the database short name, at which the file coupling begins.
4. In the "To" edit control, enter the database short name, at which the file coupling ends.
5. Press the "Insert" push button to add the defined entry to the list box.
6. Press the "Delete" push button to remove the selected entry from the list box.

For further information on physical file coupling please refer to the Adabas documentation.

Searching for DDM Fields

In large DDMs, it is often difficult to locate DDM fields. Using the search function you can flexibly search for DDM field names. If it should be necessary to replace a frequently occurring field name with another, you can use the combined search and replace function.

1. From the **Edit** menu, choose **Find**.
   Or click the **Find the specified text** toolbar button
   or press **CTRL+F**.
   The "Find" dialog box appears.
2. In the "Find Field Name" text box, enter the string to be searched for.
3. If you want the search string to be found as a whole word only and not as part of other words, select the "Match Whole Words Only" text box.
   If this box is left unselected, all occurrences of the string will be found.
4. In the "Direction" group frame, click the search direction up or down to specify whether the search will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
5. Choose **Find Next.**

If no instance of the text searched for is found, a corresponding message is displayed.

If an instance of the search string is found, it will be displayed.

To search for additional instances of the search string in the object

- From the **Edit** menu, choose **Find Next**.
  Or press **F3**.

Searching for and Replacing DDM Field Names

To search for and replace a text string in the active DDM window

1. From the **Edit** menu, choose **Replace**.
   Or click the **Replace Text** toolbar button.
   Or press **CTRL+H**.
   The "Replace Data Field" dialog box appears.
2. In the "Find Field Name" text box, enter the string to be searched for.
3. In the "Replace with" text box, enter the replacement string.
4. If the search string to be found as a whole word only and not as part of other words, select the "Match whole words only" text box.
   If this box is left unselected, all occurrences of the string will be found.
5. In the "Direction" group frame, choose the search direction up or down to specify whether the search and replace will be conducted from the cursor position to the end of the object or from the cursor position to the beginning of the object. The default is "Down".
6. Choose **Replace**.
7. Choose **Close** to exit the dialog.
   If no instance of the text searched for is found, a corresponding message is displayed.

Repeat Replace

- To do so, from the **Edit** menu, choose **Replace Next**.
  Or press **CTRL+F3**.

Replace All

With the "Replace All" function one can replace all strings at one time.

Note:
If "Replace All" is executed, all the found strings will be replaced. The "UNDO" function is not available.

Editor Window Layout

You can adapt several editor properties and functions to your own requirements. These options regulate how the editor appears and how it reacts to various types of input. For example, you can specify whether or not you want your editor window to display a status line or column header.

- Setting Editor Options
- Status Bar Information
- Column Header

Setting Editor Options

DDM Editor Options

You can set preferences for various editor options. These settings are taken as default values each time you start the DDM Editor.

To set options

- From the **Tools** menu, select **Options** and then in the options dialog select the tab "DDM Editor".

Insert before

Insert a field before the field currently selected.

Insert after

Insert a field after the field currently selected.

Status Bar

Display the status line at the top of the editor window. For information on the status line contents, see Status Bar Information.

Column Header

Display the column header at the top of the editor window. For information on the column header, see Column Header.

Short Names

Display the short name (the Adabas 2-character field name) for each field in the DDM. The short name appears under the column header SN.

If you create a new DDM field manually and the database short names are switched OFF, the editor generates a new "unused" short name for the field; this means that for this field there is no link between the database file and the DDM. Therefore, if you have to create a new DDM field, use the "Insert field" function to link the new field to the database file.

Status Bar Information

The status bar appears at the top of the window where the DDM is edited. It displays the following information:

- **Line** - The current cursor line position and the total number of lines in the DDM.
- **DBID** - The ID of the database from which the DDM is derived.
- **FNR** - The number of the file of the database from which the DDM is derived.
- **Type -** The Type of the database from which the DDM is derived.
- **Default Sequence** - The field that controls the logical sequential reading of the file when no field is specified in the READ statement of the Natural program.

Column Header

The column header line contains the following column headers for the DDM fields:

Column Header	**Description**
T	Adabas Field Type: **G** - Group **P** - Periodic group **M** - Multiple-value field **Blank** - Elementary field ***** - Comment
L	The level number assigned to the field.
SN	The 2-character field name (for ENTIRE-DB: 5-character field name). Is displayed/hidden depending on whether short names option is selected or not. For DL/I segment types, the 2-character code which is used in DL/I. For VSAM files, see the documentation Natural for VSAM.
Name	A 3 - to - 32 character field name. This is the field name used within the Natural program to reference the field. **Note:** In SQL DDMs the field name can be from 1 - to - 32 characters. DL/I: The external field name may be up to 19 characters long.
F	The format of the field. For more information on Natural formats, see the Natural Reference documentation.
Len	The standard length of the field. This length can be overridden by the user in a Natural program. For numeric fields (format N), length is specified as "*nn.m*", where "*nn*" represents the number of digits to the left of the decimal point and "*m*" represents the number of digits to the right of the decimal point. **Only for SQL DDMs:** In the length input field, you can specify either the field length as a numeric value or enter the keyword "DYNAMIC" to specify that the field length is variable. For further information see section Large and Dynamic Variables/Fields.
S	Null Value Suppression Option: **N** - Field is defined with the Adabas null value suppression option. See note at end of table. **F** - Field is defined with the Adabas fixed storage option. See note at end of table. **M** - Not null. **Blank** - Indicates no field suppression.
D	Descriptor Option: **D** - Field is an Adabas descriptor. **S** - Field is an Adabas subdescriptor or superdescriptor. **H** - Field is an Adabas hyperdescriptor. **N** - Field is an Adabas non-descriptor. **Blank** - Field is a normal field.

Note:
The value N under column header S means that null values for the field are not stored in the Adabas inverted list and will not be returned when the field is used to construct a basic search criterion (WITH clause of a FIND statement), in a HISTOGRAM statement, or in a READ LOGICAL statement.

Note:
The value F under column header S indicates that no compression is performed on the field. The field is stored according to its standard length.

Note:
For more information concerning DDMs for VSAM, please refer to the VSAM mainframe documentation.

Note:
For more information concerning DDMs for DL/I, please refer to the DL/I mainframe documentation.

Dialog Editor

The following topics are covered below:

- General Information
- The Dialog Editor Window
- Editing Dialogs
- Dialog Wizard
- Creating Dialog Elements
- Importing Data Fields
- Editing Dialog Elements
- Organizing An Application's Help File
- Setting Editor Options
- Attributes Windows for Dialogs and Dialog Elements
- Dialog Boxes

See also:

- Dialog Editor Accelerators

General Information

A single dialog is not only an isolated Natural object like a map or a program but can also represent an entire event-driven application. The dialog editor can be used to create an application with the following basic components:

- Dialog(s)
- Dialog elements
- Attributes
- Event handlers
- Data areas (local and parameter); global data areas can be referenced
- Inline subroutines

For a reference description of dialogs, dialog elements, attributes and event handlers, see the Dialog Components documentation.

For an overview of dialog editor terminology, see Introduction to Event-Driven Programming.

You can open a new dialog editor window from the Natural base window by choosing "Object > New > Dialog". Alternatively, you can edit an existing dialog by selecting it from the "Library Workspace" window.

Menus, toolbar buttons, and commands available with the dialog editor can be used to create the components of an event-driven application and edit them in various editor windows. You can create or edit another dialog, or invoke a different editor and create or edit a different type of object (for example, program, DDM or data area).

Dialog Editor Window

- Changing the Initial Position of the Dialog
- Changing the Initial Size of the Dialog
- Selecting/Deselecting Dialog Elements
- Aborting Mouse Operations
- Creation Mode in Map Editor and Dialog Editor
- Changing the Position of a Dialog Element
- Changing the Size of a Dialog Element
- Moving the Pointer
- Simulating the Mouse with the Spacebar
- Opening Windows and Dialog Boxes Using The Keyboard
- Scrolling in a Dialog
- Using the Clipboard

The dialog editor window includes a title bar, an information bar below the title bar, and a status line.

The title bar includes the name of the dialog (or "Untitled" if the dialog and the library have not been named). For example:

MYDIALOG [MYLIB] - Dialog

The information bar below the title bar contains the following information:

Item	Explanation
Status	Indicates whether the dialog has been modified since it was saved.
Selected (handle)	Indicates the handle name of the currently selected dialog element; the selection box displays the handle names of all dialog elements in the dialog together with their level number in the dialog element hierarchy. You can select another dialog element in the selection box.
x	X axis position of the currently selected dialog element relative to the upper left corner of the client area of the parent dialog element (or dialog, for top-level dialog elements). Equivalent to the current value of the RECTANGLE-X attribute.
y	Y axis position of the currently selected dialog element relative to the upper left corner of the client area of the parent dialog element (or dialog, for top-level dialog elements). Equivalent to the current value of the RECTANGLE-Y attribute.
w	Width of the currently selected dialog element. Equivalent to the current value of the RECTANGLE-W attribute.
h	Height of the currently selected dialog element. Equivalent to the current value of the RECTANGLE-H attribute.

Dialogs that are larger than the area shown in the dialog editor window can be scrolled using the scroll bars on the right and at the bottom of the dialog editor window.

Changing the Initial Position of the Dialog

To change the initial position of the dialog

1. Either click in its title bar and drag it to the desired location.
   Or open its attributes window and type in the new coordinates (in pixels) in the "X" and "Y" fields.

To open the attributes window

1. From the "Dialog" menu or from the dialog’s context menu, choose "Attributes"
   or select the dialog and press ENTER.

Changing the Initial Size of the Dialog

To change the initial size of the dialog

1. Either use the sizing border of the dialog.
   Or open its attributes window and type in the new size (in pixels) in the "W" and "H" fields.

Selecting/Deselecting Dialog Elements

It is possible to select multiple dialog elements, but only one can be active at any time. The active selection is delineated by black selection marks using which the control can be resized. The inactive selection is delineated by grey selection marks.
Dialog Editor commands which are based on a single dialog element use the active selection, whereas other (such as ’Delete’) use both the active and inactive selection. Clicking on a dialog element which is part of the inactive selection makes it the active selection without deselecting any other dialog element.

To select a dialog element

1. Click on an unselected dialog element, which becomes selected, while all other dialog elements become deselected. To select an additional dialog element, hold down SHIFT and click on the dialog element. The dialog element selected last becomes the active selection, the ones selected before are the inactive selection. To deselect the dialog element(s), click on the blank space in the dialog window.
2. Or point to the background in the dialog window and then drag the pointer to enclose or partially enclose the elements you want to select. To deselect or select additional elements, do the same as above while pressing SHIFT.
3. Or press TAB to select the next dialog element in the control sequence.
4. Or press SHIFT+TAB to select the previous dialog element in the control sequence.
5. Or select a dialog element from the drop-down list in the status bar. You can do this by using the mouse or by pressing F6 to switch to the drop-down list box and then using the arrow keys to select the dialog element. To drop down the list and view the dialog elements to choose from, press F4. To deselect the drop-down list box, press ESC or ENTER.

Note:
If one or more dialog elements are already selected, you can only additionally select other controls with the same parent.

Aborting Mouse Operations

Any operation that is completed by releasing the left mouse button may be aborted by pressing the ESC key before releasing the left mouse button.

Creation Mode in Map Editor and Dialog Editor

If you create a dialog element by selecting "Insert" plus the dialog element type, the dialog editor is in "creation mode". After creation, the dialog editor is no longer in creation mode; that is, you do not have to switch off creation mode by clicking on the dialog element as you would in the map editor.

Changing the Position of a Dialog Element

To change the position of one or more dialog elements, select the dialog element(s). The name of the dialog element selected first will be displayed in the status bar, together with its current position.

To change position, you can use one of the following options:

1. Drag the dialog element to its new location using the mouse.
2. For each selected dialog element, open its attributes window and type in the new coordinates (in pixels) in the "X" and "Y" fields.
3. Hold down SHIFT and press any arrow to move the selected dialog element(s) the number of pixels specified in the "Tools Dialog Editor Grid Settings" dialog box.
4. Hold down SHIFT+CTRL and press any arrow to move the selected dialog element(s) by one pixel.

Changing the Size of a Dialog Element

To change the size of one or more dialog elements, select the dialog element(s). You then have the following options:

1. Point to one of the eight small black squares (the selection mark of the last selected dialog element). The mouse pointer now indicates the direction into which you can resize the dialog element. Hold down the left mouse button and drag (one of) the dialog element(s) to the desired size. If more than one dialog element is selected, the other dialog elements selected are resized proportionally.
2. Open the dialog element’s attributes window and type in the new size (in pixels) in the "W" and "H" fields.
3. Choose "Control > Stretch", then the direction into which you can resize the dialog element. Then use the mouse or the keyboard to continue the operation.

Moving the Pointer

To move the pointer, you have three options:

1. Move the mouse; or
2. Press any arrow key to move the pointer by the number of pixels specified in the "Options Dialog Editor Grid Settings" dialog box; or
3. Hold down CTRL and press any arrow key to move the pointer by one pixel.

Simulating the Mouse with the Spacebar

You can simulate mouse operations with the spacebar as described in the following table. Note that the pointer must lie on the element to be manipulated.

Mouse Operation	Keyboard Operation
Press left mouse button	Press and hold down the spacebar.
Release left mouse button	Release the spacebar.
Click mouse	Press and release the spacebar.
Double-click mouse	Press and release the spacebar twice.
Move dialog element	Move pointer to element, press and hold down the spacebar, press the appropriate arrow key(s).
Select several dialog elements	Move pointer to background, press and hold down the spacebar, press the appropriate arrow key(s).
Resize dialog element	Move pointer to any black square of selected element, press and hold down the spacebar, press the appropriate arrow key(s), release the spacebar.

Simulating a mouse double-click with the spacebar opens the attributes window for the dialog element on which the pointer is positioned; if the pointer is not positioned on any dialog element, the dialog attributes window is opened.

Opening Windows and Dialog Boxes Using The Keyboard

Key (Combination)	Opens
ENTER	Attributes window of the selected dialog element or of the dialog, if no dialog element is selected.
CTRL+ALT+E	"Event Handlers" dialog box for the dialog.
CTRL+SHIFT+E	"Event Handlers" dialog box for the selected dialog element.
SHIFT+ENTER	"Event Handlers" dialog box for the selected dialog element or for the dialog, if no dialog element is selected.
CTRL+ALT+S	The dialog’s "Subroutines" dialog box.
CTRL+ALT+M	Menu bar attributes window.
CTRL+ALT+T	Toolbar attributes window.
CTRL+ALT+I	Timer attributes window.
CTRL+ALT+L	The dialog’s "Local Data Area" dialog box.
CTRL+ALT+P	The dialog’s "Parameter Data Area" dialog box.
CTRL+ALT+G	The dialog’s "Global Data Area" dialog box.

Scrolling in a Dialog

You can scroll in a dialog window if at least one dialog element is outside its scroll range. For you to be able to scroll in a dialog, the dialog scroll bars must be active. To activate the dialog scroll bars, open the dialog attributes window either by pressing ENTER or by double-clicking in the dialog. Then click either the "Horizontal Scrollbar" or "Vertical Scrollbar" entry.

To scroll with the mouse, you either:

1. Point to the scroll-bar slider and drag the slider in the desired direction; or
2. Point to the scroll-bar shaft and click; or
3. Point to one of the scroll-bar arrow buttons and hold down the left mouse button.

To scroll with the keyboard, you do not need a scroll bar. You have four options:

1. To simulate clicking into a vertical scroll bar, press the PAGE UP or PAGE DOWN keys; or
2. To simulate clicking into a horizontal scroll bar, press SHIFT+PAGE UP or SHIFT+PAGE DOWN; or
3. To simulate clicking on the corresponding vertical arrow button, press CTRL+PAGE UP or CTRL+PAGE DOWN; or
4. To simulate clicking on the corresponding horizontal arrow button, press CTRL+SHIFT+PAGE UP or CTRL+SHIFT+PAGE DOWN.

Using the Clipboard

Key (Combination)	Function
DEL	Delete the selected dialog element
CTRL+C	Copy
CTRL+V	Paste

Editing Dialogs

- Editing a Dialog’s Source Code
- Editing a Dialog’s Attributes
- Editing a Dialog’s Event Handlers
- Defining a Dialog’s Menu Bar
- Defining a Dialog’s Toolbar
- Creating and Maintaining Timers for a Dialog
- Adding a Comment Section to a Dialog
- Defining a Parameter or Local Data Area for a Dialog
- Selecting a Global Data Area for a Dialog
- Defining an Inline Subroutine for a Dialog
- Defining the Control Sequence in a Dialog

Editing a Dialog’s Source Code

To edit a dialog’s source code

1. Load the dialog into the editor.
2. From the "Dialog" menu, choose "Source Code...".
   Or press CTRL+ALT+C.
   The dialog’s source code window appears and the program editor is loaded. This editor enables you to scan for text strings, replace them, and so on. For more information on how to use the program editor, see The Program Editor.
   You can switch between the dialog editor and the program editor by selecting the source code window or the dialog window. If you edit in either window, you need to synchronize your updates: (graphically) modifying the dialog locks the source code window and you may not make changes there. Correspondingly, if you change the source code, you may not make changes in the dialog window, which is locked. If your editor is locked, its status bar displays "Locked".
   If a source code window is open, but not active, you can activate it by choosing "Source Code..." from the "Dialog" menu.
   When you issue a command from the program editor window that affects the source code, such as "Save" or "Run", the dialog editor updates itself automatically by scanning the source code, displaying the modified dialog, and then regenerating the source code.When you issue a command from the dialog editor window after you have modified the code in the source code window, you are prompted whether you want to update the source code or not.
3. To stow any modified source code: from the program editor’s "Object" menu, choose "Stow".

Whenever you want to save a dialog under a new name, select "Save as" from the "Object" menu, where for the "Save dialog as" dialog box appears.

Editing a Dialog’s Attributes

To edit a dialog’s attributes

1. Load the dialog into the editor.
2. From the "Dialog" menu or from the dialog’s context menu, choose "Attributes...".
   Or double-click on the dialog.
   Or press ENTER.
   The dialog’s attributes window appears. To find out what the entries in the attributes window mean, choose "Help". For context-sensitive help, select the attribute entry and press F1.
3. Enter the desired attribute values.
4. Choose OK to confirm your changes.

Editing a Dialog’s Event Handlers

To edit a dialog’s event handlers

1. Load the dialog into the editor.
2. From the "Dialog" menu or from the dialog’s context menu, choose "Event handlers...".
   Or press CTRL+ALT+E or SHIFT+ENTER.
   The dialog’s event handler section appears.
3. Select the type of event (such as BEFORE-OPEN or ERROR).
   Or choose "New" to enter a user-defined event.
   Or choose "Rename" to save a user-defined event with a new name.
4. Enter the desired event code in free form either in the edit window in the "Dialog Event Handler"window itself, or using the Program Editor. To use the Program Editor, click the "Editor" push button, then close the "Dialog Event Handler" window using the "OK" push button. This code will be executed when the event occurs for the dialog. Note that if you have specified code in the before-any and after-any event sections, this will be triggered before and after the code entered here. So if you need common event code, you only have to enter it once in your dialog’s before-any and after-any event section.
5. Choose OK to save your code, if using the "Dialog Event Handler" window. If using the Program Editor, the code can be saved by choosing "Save" from the "Object" menu, or by closing the Program Editor and electing to save the changes when prompted.

Defining a Dialog’s Menu Bar

▶ **To define a dialog’s menu bar**

1. Load the dialog into the editor.
2. From the "Dialog" menu, choose "Menu bar...".
   Or press CTRL+ALT+M.
   A dialog box appears asking you whether you want to turn the dialog’s menu bar setting on.
3. Choose "Yes".
   A blank default menu bar is added to the dialog and the menu bar’s attributes window appears. For more information on the attributes window, see the section Menu Editor Window.
4. Choose OK to confirm your changes.

Defining a Dialog’s Toolbar

▶ **To define a dialog’s toolbar**

1. Load the dialog into the editor.
2. From the "Dialog" menu, choose "Toolbar...".
   Or press CTRL+ALT+T.
   A dialog box appears asking you whether you want to turn the dialog’s toolbar setting on.
3. Choose "Yes".
   A blank default toolbar is added to the dialog and the toolbar’s attributes window appears. For more information on the attributes window, see the section Toolbar Control Attributes Window. In this section you will also find information on new toolbar control features.
4. Choose OK to confirm your changes.

Creating and Maintaining Timers for a Dialog

You use timers to trigger a dialog event periodically.

▶ **To create and maintain a timer for a dialog**

1. Load the dialog into the editor.
2. From the "Dialog" menu, choose "Timers...".
   Or press CTRL+ALT+I.
   The timer’s attributes window appears. For more information on the attributes window, see the section Timer Attributes Window.
3. Choose OK to confirm your changes.

Adding a Comment Section to a Dialog

To add a comment section to a dialog

1. From the "Dialog" menu, choose "Dialog comment...".
   Or press CTRL+ALT+O.
   The dialog comment section appears where you can enter your comments in free form. Please note that you do not have to use the "/*" notation when entering comments in the text area. If you are listing your dialog code, you will find your comment at the beginning.
2. Choose OK to save your comment.

Defining a Parameter or Local Data Area for a Dialog

To define a local data area for a dialog

1. From the "Dialog" menu or from the dialog's context menu, choose "Local Data Area...".
   Or press CTRL+ALT+L.
   The definition section for the local data area appears. In a local data area, you must include all the user-defined variables or other variables that you want to use in an event handler code section or a subroutine of the current dialog. Note that the dialog editor automatically generates the data definitions for the dialog elements.
   The "Using" button opens a dialog box that enables you to include existing inline data definitions.
2. Choose OK to save your data definition.

To define a parameter data area for a dialog

1. From the "Dialog" menu, choose "Parameter Data Area...".
   Or press CTRL+ALT+P.
   The definition section for the parameter data area appears. In a parameter data area, you must include all the parameters that you want to be passed on to the current dialog in an OPEN DIALOG or SEND EVENT statement.
   The "Using" button opens a dialog box that enables you to include existing inline data definitions.
2. Choose OK to save your data definition.

Selecting a Global Data Area for a Dialog

To select a global data area for a dialog

1. From the "Dialog" menu, choose "Global Data Area...".
   Or press CTRL+ALT+G.
   A dialog box appears where you can select a global data area for the dialog.
2. Select an entry in the "Available Global Data Areas" list box.
3. Choose OK.

Defining an Inline Subroutine for a Dialog

To define an inline subroutine for a dialog

1. Load the dialog into the editor.
2. From the "Dialog" menu or from the dialog’s context menu, choose "Inline Subroutines...".
   Or press CTRL+ALT+S.
   The "Dialog inline subroutines" code section appears.
3. Choose "New" to enter a new subroutine.
   Or select the name of an existing subroutine you want to edit.
   If you have chosen "New", a dialog box prompts you for a name.
4. Enter the name of the new subroutine.
5. Choose OK.
6. Enter the desired subroutine code in free form, either directly in the window itself or using the Program Editor by clicking the ’Editor’ push button, then close the window using the’OK’ push button.
7. Choose OK to save your code.

Defining the Control Sequence in a Dialog

The control sequence is the keyboard navigation sequence in which the end user will go through the dialog elements.

To define the control sequence of a dialog

1. From the "Dialog" menu, choose "Control sequence", or press CTRL+ALT+Q
   The control sequence is displayed as a number at the top left corner of each dialog element. The editor is now in navigation sequence definition mode.
2. Use the mouse to select the dialog elements in the desired sequence.

If you *do not* select a dialog element before enabling navigation sequence definition mode, the next dialog element that you select will be the first in the navigation sequence. Its number is greyed and you can select the next dialog element in the sequence, and so on.

If you *do* select a dialog element, you can redefine the sequence from this element onwards.You can also select a dialog element when in control sequence editing mode without resequencing it by holding down the SHIFT key whilst making the selection. This is especially useful if the selected dialog element is one of the last elements in the sequence - you do not have to redefine the sequence of all preceding dialog elements. Note that instead of selecting each dialog element with the mouse, you can also select them from the selection box in the status bar of the dialog editor. This selection box always shows the dialog elements in their control sequence.

You can exit control sequence editing mode implicitly, by selecting another command (e.g. ’Insert Push Button’) or explicitly by selecting the ’control sequence’ menu of this again, or simply by pressing ESC.

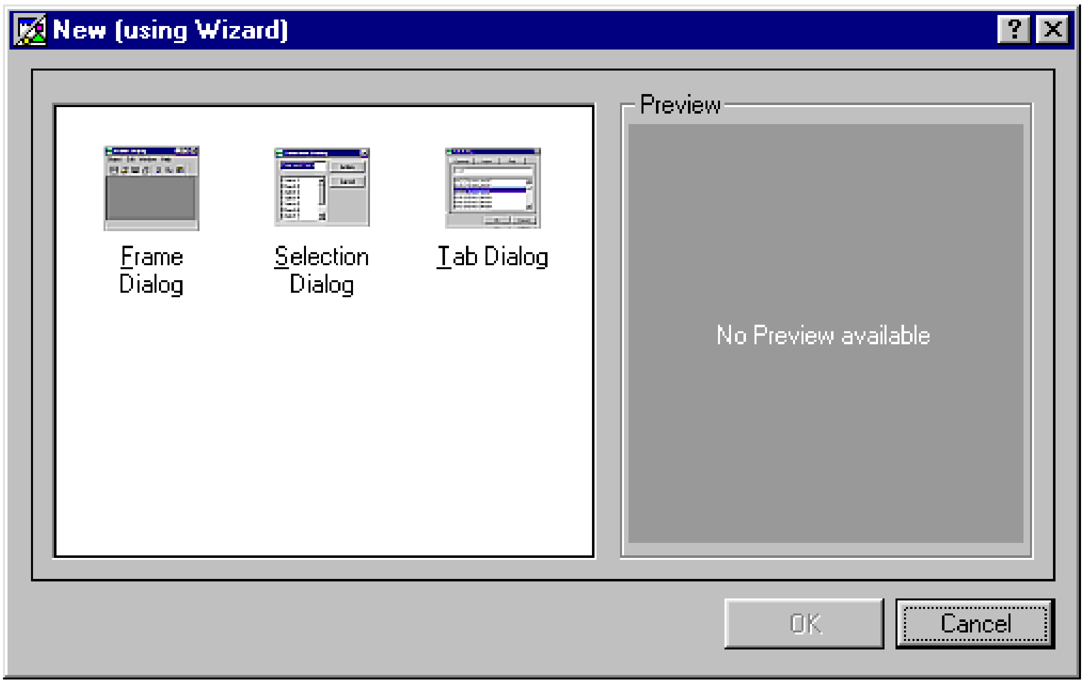
Note:
The control sequence also decides the order in which the dialog elements overlap.

Dialog Wizard

The Dialog Wizard is a tool for creating dialogs for specific purposes. The defined dialogs can have several layouts that adapt to desired requirements.

The generated dialog can be modified with the dialog editor. In the dialog there are User Code Sections with sample coding for data retrieval and result flagging. These sections should be replaced by user-specific requirements.

To activate the Wizard:



- From the "Menu Bar" select "Object" > "New" > "Dialog Wizard".

Frame Dialog

In the Frame Dialog Wizard a new dialog can be created in a Frame Layout. The structure of the Frame Dialog for example is applicable in an application frame.

A default Frame Dialog is generated if you define nothing in the wizard.

Selection Dialog

In the Selection Dialog Wizard a new dialog can be created in a Selection Layout. The structure of a Selection Dialog for example is applicable for reading, saving or opening objects.

A default Selection Dialog is generated if you define nothing in the wizard.

Tab Dialog

In the Tab Dialog Wizard a new dialog can be created in a Tab Layout. The structure of a Tab Dialog for example is applicable in a help dialog or for option settings.

A default Tab Dialog is generated if you define nothing in the wizard.

Creating Dialog Elements

To create a dialog element

1. From the "Insert" menu, choose one of the following entries, depending on which dialog element type you wish to create:
   "ActiveX control","Bitmap", "Canvas", "Control Box", "Edit area", "Group frame", "Input field", "List box", "OLE container", "Push button", "Radio button", "Scroll bar", "Selection box", "Table", "Text constant", "Toggle button".
   After one of these items has been selected, you are in creation mode. If you move the mouse within the dialog window, the cursor shape is a cross with a minimized graphical representation of the dialog element to be created.
2. Move the cursor to the desired upper left position of the dialog element.
3. Either hold down the left mouse button, drag the cursor until you have created the desired outline of the new dialog element and release the mouse button.
   Or click or press ENTER.
   This creates a dialog element with a default size.

The control sequence is the keyboard navigation sequence in which the end user will go through the dialog elements. It is decided by the order in which you create the dialog elements. When you create a new dialog element, it is inserted after the active selection and any of its successive direct and indirect children if the active selection shares the same parent as the newly-inserted control. If not, the insertion point is based on the last dialog element with the same parent which precedes the active selection in the control sequence. If there is no such control, or if no controls are selected, the new control is inserted immediately before the first control with the same parent, or immediately after its container if no such control exists. You can modify this default sequence by choosing "Dialog > Control Sequence". For more information, see the section Defining the Control Sequence in a Dialog.

Note:
The same rules apply to dialog elements created by pasting them from the clipboard.

If you insert a new dialog element dynamically by using the PROCESS GUI statement action ADD, you decide its position in the navigation sequence by creating the dialog element and setting the SUCCESSOR attribute to the handle value of its successor.

Importing Data Fields

You can import a data field from an object in your Natural environment into an input field control or a selection box control which you create at the same time.

To import data fields

1. From the "Insert" menu, choose "Import".
2. From the "Import" submenu, choose "Input field" or "Selection box".
   The "Import Data Field" dialog box appears.
3. Enter the library and the Natural object type.
   A list of objects appears.
4. Choose an object.
   A list of data fields appears.
5. Choose the data field(s) you want to use for creating a dialog element.
6. Choose "Import".

The selection box control or the input field control is created with the selected data field(s). Note that the fields themselves are not imported.

Editing Dialog Elements

- Cutting a Dialog Element
- Copying a Dialog Element
- Pasting a Dialog Element from the Clipboard
- Deleting a Dialog Element
- Selecting all Dialog Elements with the same Parent in a Dialog
- Editing a Dialog Element's Attributes
- Editing a Dialog Element's Event Handlers
- Unifying the Size of Several Dialog Elements
- Aligning the Position of Several Dialog Elements
- Unifying the Spacing Between Several Dialog Elements
- Stretching a Dialog Element

To edit one or several dialog elements as a whole, you can use the entries provided in the "Edit menu". These entries enable you to reuse dialog elements for similar contexts instead of creating them from scratch.

Cutting a Dialog Element

To cut a dialog element

1. Select the dialog element.
2. From the "Edit" menu or from the dialog element's context menu, choose "Cut".
   Or click the "Cut" toolbar button.
   Or press SHIFT+DEL or CTRL+X.
   The selected dialog element and any of its child dialog elements is cut to the clipboard for pasting elsewhere, for example into other dialogs.

Note:
You can also select several or all dialog elements in a dialog and cut them all at once.

Copying a Dialog Element

To copy a dialog element

1. Select the dialog element.
2. From the "Edit" menu or from the dialog element's context menu, choose "Copy".
   Or click the "Copy" toolbar button.
   Or press CTRL+INS or CTRL+C.

The selected dialog element is copied to the clipboard for pasting elsewhere. If the selected dialog element has child dialog elements you will be prompted as to whether these should also be copied or not.

Note:
You can also select several or all dialog elements in a dialog and copy them all at once.

Pasting a Dialog Element from the Clipboard

To paste a dialog element from the clipboard

- From the "Edit" menu or from the dialog element's context menu, choose "Paste".
  Or click the "Paste" toolbar button.
  Or press SHIFT+INS or CTRL+V.

The dialog element in the clipboard is pasted into the current container in the current dialog. The current container is the lowest level dialog element containing the selected dialog elements which is not the selected dialog element itself. If you are pasting a dialog element back into the same container from which it was copied, the original dialog element is overlaid by the copy. You then have to move the pasted dialog element to its new location. (The pasted dialog element is preselected by default.) Note that if it is desired to paste dialog elements into an empty container, a dummy child dialog element must be created and selected first.

Note:
You can paste several or all dialog elements in a dialog if you have cut or copied them to the clipboard at once.

Deleting a Dialog Element

To delete a dialog element

1. Select the dialog element.
2. From the "Edit" menu or from the dialog element’s context menu, choose "Delete".
   Or click the "Delete" toolbar button.
   Or press DEL.
   A dialog box appears asking you to confirm the deletion.
3. Choose "Yes".

The selected dialog element is deleted, together with any of its child dialog elements.

Note:
You can also select several or all dialog elements in a dialog and delete them all at once.

Selecting all Dialog Elements with the same Parent in a Dialog

To select all dialog elements in a dialog

- From the "Edit" menu or from the dialog element’s context menu, choose "Select all".
  If a dialog element is selected, all unselected dialog elements with the same parent become selected. If no dialog element is selected, all top-level dialog elements become selected.

Editing a Dialog Element’s Attributes

To edit a dialog element’s attributes

1. Select the dialog element.
2. From the "Control" menu or from the dialog element’s context menu, choose "Attributes...".
   Or double-click on the dialog element.
   Or press ENTER.
   The dialog element’s attributes window appears. To find out what the entries in the attributes window mean, choose "Help". For context-sensitive help, select the attribute entry and press F1.
3. Enter the desired attribute values.
4. Choose OK to confirm your changes.

Editing a Dialog Element’s Event Handlers

To edit a dialog element’s event handlers

1. Select the dialog element.
2. From the "Control" menu or from the dialog element’s context menu, choose "Event handlers...".
   Or press SHIFT+ENTER.
   The dialog element’s event handler section appears.
3. Select the event (such as click or double-click).
4. To view the event parameters of an ActiveX control, click the "Event Info..." button.
5. Enter the desired event code in free form, either directly in the window itself or by using the Program Editor by clicking the ’Editor’ push button, then closing the window with the ’OK’ push button.
   Or choose "Use" to enter a user-defined event.
   This code will be executed when the event occurs for the dialog element. Note that if you have specified code in the before-any and after-any event sections, this will be triggered before and after the code entered in Step 4. So if you need common event code, you only have to enter it once in your dialog’s before-any and after-any event section.
6. Choose OK to save your code.

Unifying the Size of Several Dialog Elements

To unify the size of several dialog elements

1. Select all dialog elements whose size is to be unified.
2. Select the dialog element that has the reference width or height.
3. From the "Control" menu, choose "Unify size".
4. From the "Unify size" submenu, choose "Width".Or choose "Height".
   The dialog elements are aligned to the width or height of the reference dialog element.

Aligning the Position of Several Dialog Elements

To align the position of several dialog elements

1. Select all dialog elements whose position is to be aligned.
2. Select the dialog element that has the reference position.
3. From the "Control" menu, choose "Align position".
4. From the "Align position" submenu, choose "Left".
   Or choose "Center (horizontal)".
   Or choose "Right".
   Or choose "Top".
   Or choose "Center (vertical)".
   Or choose "Bottom".

The dialog elements are aligned to the position of the reference dialog element.

Unifying the Spacing Between Several Dialog Elements

To unify the spacing between several dialog elements

1. Select all dialog elements between which you want to unify spacing.
2. From the "Control" menu, choose "Unify spacing".
3. From the "Unify spacing" submenu, choose "Horizontal".
   Or choose "Vertical".
   The spaces between the dialog elements are now distributed evenly.

Stretching a Dialog Element

To stretch a dialog element in a particular direction

1. Select the dialog element to be stretched.
2. From the "Control" menu, choose "Stretch".
3. From the "Stretch" submenu, choose a direction, for example "North west".
   A cursor appears indicating that you can stretch the dialog element.
4. Drag the cursor with the mouse until your dialog element has the desired size.
5. Click with the left mouse button to fix the dialog element’s size.
   The dialog element is now resized; as it is still selected, you can edit it further.

Note:
You can also select several dialog elements and stretch them all at the same time.

Organizing An Application’s Help File

- Using The Help Organizer’s Main Dialog
- Generating Help IDs
- Extending A Help ID Definition
- Editing The Global Topic List

Help files provide online information about an application’s functions.

Whenever an end user must be given general information on how your application is structured, a help text is needed that can be invoked from within the application.

Whenever an end user must be given specific information on what may be entered in an input field control, for example, a short text is needed that can be accessed from the input field control.

To keep an overview of all the different help sections in an application, Natural provides you with the help organizer. With this organizer,

- you assign a help ID (HELP-ID attribute value) to a specific dialog element;
- you write the help text for this help topic; this text is converted to a .rtf file to be processed by the help compiler;
- you optionally define the help topic’s keywords;
- you optionally assign a help compiler macro to the help topic;
- and you optionally add a comment for your internal documentation purposes.

It is also possible to use several help files for one application. In this case, you must specify the help file in the dialog’s attributes window. You can also use the same help file for more than one dialog.

The .rtf file and the corresponding help topic information created with the help organizer must be converted to a .hlp file. In this way, your end user can retrieve help information from the Natural application. This conversion into a .hlp file is done with the help compiler.

The Natural installation procedure also installs a version of the Microsoft help compiler on your disk. You will find the help compiler in a subdirectory of *drive-letter*:\SAG\*product-acronym**version-number*\Natural which is called HC*n* where *n* depends on the operating system on which Natural has been installed.

For more information on how to use the help compiler and on how to create the .hlp file from your .rtf file, please refer to the READ*n*.TXT file in this directory where *n* is depending on the operating system.

There is also an example of a small help file which was generated with the help organizer.

This example help file is called BROWHLP.HLP and is located in the directory referred to by the environment variable NATGUI_BMP. You can find the files BROWHLP.RTF, BROWHLP.HPJ and README.TXT in the following directory:

drive-letter:\SAG\*product-acronym**version-number*\Natural\SAMPLES\HELP\

To see a demonstration of the example help file

1. Go to library SYSEXEVT.
   This library contains a browse application with a dialog called BROWSE1.
2. Start the SAG-DEMO-DB database. This is a prerequisite to running the browse application.
3. Start dialog BROWSE1 to run the application.
4. Press F1 with the focus being on various elements of the application.

To see a demonstration of the help organizer (looking at the contents of BROWHLP.RTF)

- Follow the instructions in the README.TXT file.

For each help topic you create, you can in general follow this sequence of steps

1. Invoke the help organizer’s main dialog.
2. Select a dialog element.
3. Generate a new help topic ID or enter the ID of your choice. To generate a help topic ID automatically, the help ID must be "0" (default). To enter a help topic ID, either you fill in the "Help ID" entry of the dialog or dialog element, or you fill in the help topic ID in the "Online Help Organizer" main dialog.
4. Return to the "Online Help Organizer" dialog.
5. Assign the help topic ID.
6. Enter the external definitions for the help topic ID, such as the help text and the topic.
7. Return to the "Online Help Organizer" dialog.
8. Go to the topic list and see whether this new help topic fits your general organization of the help file to be created.
9. Return to the "Online Help Organizer" dialog.
10. Save everything.

These steps are described in detail in the following sections.

Using The Help Organizer’s Main Dialog

▶ To invoke the "Online Help Organizer" dialog

- From the "Dialog" menu, choose "Help organizer..."
  Or press CTRL+ALT+H.

The main dialog appears. It contains the following:

Entry	**Meaning**
Caption	Online Help Organizer - *pathname* (location of the .rtf file to be generated from a help topic text).
Help IDs in Dialog	Name of the current dialog. You organize your Help IDs dialog by dialog.
Control Name	The handle name or the handle names of all dialog elements inside the dialog. You can not change these handle names here.
Help ID	The Help ID attribute value for the dialog or dialog element. You may assign several Help ID values to one dialog element. If there are several Help IDs, you can view them by pulling down the drop-down list box. If you select a Help ID, either you can extend its definition or you can enter a new Help ID of your choice.
Topic	The name of the help topic. You can not edit this name here, but in the help topic’s extended definition dialog box.
Help text	The help text. You can edit this text here or in the help topic’s Extended Definition dialog box. **Note:** It is possible to enter RTF formatting commands if the help text begins with a "{\" and ends with a "}". This signifies "raw mode". In raw mode, special RTF characters (’{’, ’}’ and ’\’) must be escaped by a preceding backslash if they are to be treated as literal characters. However, even in raw mode, it is not necessary to enter line feed (\line) commands explicitly.
Action:	
New ID...	Invokes the "Help ID Generation" dialog where you can generate a Help ID for those dialogs or dialog elements for which the Help ID is still "0" (default).
Ext Def...	Invokes the "Extended Definitions - (...)" dialog.
Clear	Resets the currently selected Help ID to zero and clears the help topic text that was associated to the Help ID.
Topic List	Invokes the "Global Topic List" dialog.
OK	Saves your settings, exits the dialog and generates the necessary help files. These files include a "*helpfilename*.rtf" help topic file, a "*helpfilename*.hpj" help project file, a "*helpfilename*.cnt" contents file, a "*helpfilename*.hm" Help ID mapping file and a "*helpfilename*.cshelp.cnt" contents include file. The help project and contents files are only generated if they do not already exist, ensuring that any changes made directly to these files (e.g. via a text editor) are not lost.
Apply	Saves your settings and generates the help files (as for the OK pushbutton) without exiting the dialog. Allows you to save the changes and switch to and from the help compiler without having to leave and re-enter the help organizer.
Cancel	Exits the dialog without saving the settings.
Help	Provides online help on the dialog.

The Online Help Organizer dialog may be re-sized. The position and dimensions are saved between sessions in the user profile (.PRU).

Generating Help IDs

To generate a new help topic ID

1. From the "Help Organizer Main" dialog, select a dialog or dialog element and choose the "New ID" button. The "Help ID Generation" dialog appears.
2. (Optional) Enter a starting value which Natural will use to generate the next new ID. This Help ID is valid only for the current dialog. To avoid duplicate Help IDs in the same Natural application, you define exclusive numerical ranges for the help topic IDs of a dialog. The starting value enables you start off the Help IDs at the beginning of the numerical range. You can, for example, decide that your dialog uses the range of 50 to 60. You enter "50" as the starting base and your first generated ID will be 50.
3. Choose OK to generate the Help ID. Or choose "Cancel" to exit the dialog without generating.

Note:
You can also generate Help IDs for several dialogs/dialog elements at once. To do so, you must select several dialogs/dialog elements in the "Online Help Organizer" dialog. The IDs are then generated sequentially starting from the lowest ID available.

Extending A Help ID Definition

To extend a help topic ID's definition

- From the "Help Organizer Main" dialog, select a Help ID and choose the "Ext Def" button.

The "Extended Definitions" dialog appears. It contains the following:

Entry	**Meaning**
Caption	Extended Definitions - [*current HELP-ID*]
Topic:	Enter the heading of the help topic here.
Help Text:	Enter the text of the help topic here.
Keywords:	Enter the keywords of your help topic here. Keywords enable the end user of your Natural application to find the topic by selecting the "Search" button in the "Help" window.
Browse sequences:	Enter your browse sequences here. A browse sequence is useful if you want to group your help topics by subject. Browse sequences can also consist of an optional alphanumeric sort key, separated from the browse sequence name by a colon (e.g. *mysequence:sort_key*).
Macro:	Enter the name of your help macro here. Help macros enable you to customize your help.
Comment:	This section is for your internal help project documentation.
OK	Saves your settings and exits the dialog.
Cancel	Exits the dialog without saving the settings.
Help	Provides online help on the dialog.

This information will be part of the .rtf file to be generated and will be interpreted by the help compiler.

The Extended Definitions dialog may be resized. The position and dimensions are saved between sessions in the user profile (.PRU).

Editing The Global Topic List

To edit the global topic list

- From the "Help Organizer Main" dialog, choose the "Topic List" button.

The "Global Topic List" dialog appears. It contains the following:

Entry	Meaning
Caption	Global Topic List - [File: *selectedfilepath*]
Line in the table	Each line in the table represents a help topic. It lists its Help ID, the topic name and the extended definitions associated with it.
Action:	
Undo	Undoes the last editing action, especially deletion.
Delete	If a help topic is selected, this topic is deleted.
Ext Def...	Invokes the "Extended Definitions" dialog for the selected help topic.
Move Entry	Allows you to change the sequence of help topic entries.
MoveUp	Moves the selected help topic upwards within a browse sequence of help topic entries.
MoveDown	Moves the selected help topic downwards within a browse sequence of help topic entries.
OK	Saves your settings and exits the dialog.
Cancel	Exits the dialog without saving the settings.
Help	Provides online help on the dialog.

This list contains all of the topics in the help file. Note that the topics are always maintained in browse sequence order.

The Global Topic List dialog may be resized. The position and dimensions are saved between sessions in the user profile (.PRU).

Setting Editor Options

You can set preferences for various editor options. These settings are taken as default values each time you start the Dialog Editor.

The following options are available:

- Enabling The Enhanced Listing Option
- Displaying Or Hiding The Status Bar
- Turning The Crosshair Cursor On And Off
- Turning Autoscroll On And Off
- Displaying The Dialog Inside Or Outside The Editor
- Displaying Bitmaps
- Displaying Or Hiding The Grid
- Customizing The Grid
- Saving Editor Options With A Particular Dialog

You can adapt several editor options to your personal requirements. These options regulate how the editor appears and reacts to various types of input. For example, you can specify whether or not you want your editor window to display a status bar.

Enabling The Enhanced Listing Option

The enhanced listing option of a dialog enables you to view the dialog source in a format that is more readable than the traditional Natural source code. If you choose to display the code listing after receiving a compiler error, you will get the code generated for the compiler. By default, the dialog editor enables enhanced list mode.

Note:
Enabling this option is only meaningful if you list or print "non-enhanced" dialog sources, not for the new dialog source format. Note also that if you have saved your dialog sources in non-enhanced format and the enhanced listing option is enabled, any Natural runtime error message will contain an incorrect line number. To ensure that you get the correct line number, disable the enhanced listing option. As under Natural for Windows and Unix/OpenVMS Version 4.1, dialog sources are always saved in enhanced format, this line number inconsistency does not exist, and the older ('22C') format is only seen if old dialogs in this format are listed or printed outside of the dialog editor.

To view the enhanced listing

- From the "Object" menu, choose "List".

To enable enhanced dialog list mode

1. From the "Options" dialog, check the "Dialog Editor" tab.
2. Check the "Enhanced dialog list mode" toggle button.

Displaying Or Hiding The Status Bar

By default, the dialog editor displays the status bar.

▶ **To display or hide the status bar**

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Status Bar" toggle button.

Either the toggle button now has a check mark to indicate display, or the check mark is no longer displayed to indicate hiding.

Turning The Crosshair Cursor On And Off

By default, the dialog editor displays the system cursor. If the crosshair is turned on, and you create or move dialog elements, lines appear that extend to the window borders.

▶ **To display or hide the cross-hair cursor**

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Crosshair" toggle button.

Either the toggle button now has a check mark to indicate display, or the check mark is no longer displayed to indicate hiding.

Turning Autoscroll On And Off

If the toggle button is checked, the Dialog Editor window is automatically scrolled to make the dialog visible if the dialog is moved wholly or partially outside the current visible range.

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Autoscroll" toggle button.

Either the toggle button now has a check mark to indicate display, or the check mark is no longer displayed to indicate hiding.

Displaying The Dialog Inside Or Outside The Editor

The dialog editor allows you to specify whether the current dialog is displayed inside the dialog editor window (default). Displaying the dialog outside enables more space for editing. Please be careful when setting this option to "off". Your dialog may disappear from view if its RECTANGLE-X and RECTANGLE-Y values denote a position outside the screen. Another consequence is that the dialog will overlap any code listing window that appears.

To display the dialog inside or outside the editor

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Display dialog inside editor" toggle button.

Either the toggle button now has a check mark to indicate display inside, or the check mark is no longer displayed to indicate display outside.

Displaying Bitmaps

By default, the dialog editor loads and displays the bitmaps in the dialog elements to show how they appear at runtime. If you switch this option off, the dialog element displays the bitmap name only, which improves editing performance.

To turn the bitmap display on or off

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Display bitmaps" toggle button.

Either the check box now has a check mark to indicate display, or the check mark is no longer displayed to indicate unloading of bitmaps.

Displaying Or Hiding The Grid

The dialog editor enables you to display a grid to help align dialog elements. By default, the grid is not displayed.

To display the grid

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check or uncheck the "Display Grid" toggle button.

Either the check box now has a check mark to indicate display, or the check mark is no longer displayed to indicate hiding.

Customizing The Grid

▶ **To customize the grid**

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Choose the "Lines" option button (which will result in a classical grid).
   Or choose the "Dots" option button (which will result in a dots pattern).
3. Choose the "Color" command button.
   The operating system’s "Color" dialog box is displayed.
4. Select a color.
   Or define a custom color.
5. Choose OK. A sample rectangle of the chosen colour is displayed.
6. For the horizontal grid (X) axis, enter the number of pixels (steps) between two lines or dots.
7. For the x axis, enter a numerical starting value for the grid.
8. Choose the "Snap to Grid" check box if you want the dialog elements to snap to the grid.
9. Repeat Steps 7 to 9 for the y axis.
10. Choose OK to save the grid settings and return to the dialog editor.

Saving Editor Options With A Particular Dialog

The dialog editor enables you to save the option settings with the dialog in order to customize editor usage to individual dialogs.

▶ **To save the editor options with the current dialog**

1. From the "Options" dialog, choose "Dialog Editor" tab.
2. Check the "Save settings with dialog..." check box.

The options will be saved together with the dialog itself whenever any dialog is saved. Dialogs saved whilst this option was inactive will continue to use the current option settings.

Attributes Windows for Dialogs and Dialog Elements

This section explains the editing options in the attributes windows for dialogs and dialog elements.

- ActiveX Control Attributes Window
- ActiveX Control Property Pages
- Bitmap Control Attributes Window
- Canvas Control Attributes Window
- Control Box Control Attributes Window
- Dialog Attributes Window
- Dialog Context Menus Window
- Edit Area Control Attributes Window
- Group Frame Control Attributes Window
- Input Field Control Attributes Window
- List Box Control Attributes Window
- Menu Editor Window
- OLE Container Control Attributes Window
- Push Button Control Attributes Window
- Radio Button Control Attributes Window
- Scrollbar Control Attributes Window
- Selection Box Control Attributes Window
- Signal Attributes Window
- Status Bar Control Attributes Window
- Status Bar Control Attributes Subwindow
- Table Attributes Window
- Text Constant Control Attributes Window
- Timer Attributes Window
- Toggle Button Control Attributes Window
- Toolbar Attributes Window
- Toolbar Control Attributes Window
- Toolbar Control Attributes Subwindow

ActiveX Control Attributes Window

Accessible Using

1. Double-click on the ActiveX control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	**Represents**
Name	Handle name of the ActiveX control (may be overwritten with another name).
Control	Name of the ActiveX control.
DIL Text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Style:	
OK Button	STYLE attribute value: if the end user presses ENTER, this button is pushed. This attribute is only available for ActiveX controls that behave like buttons. These ActiveX controls are marked with the style OLEMISC_ACTSLIKEBUTTON in the system registry.
Cancel Button	STYLE attribute value: if the end user presses ESC, this button is pushed. This attribute is only available for ActiveX controls that behave like buttons. These ActiveX controls are marked with the style OLEMISC_ACTSLIKEBUTTON in the system registry.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Rectangle:	The following four attributes decide the ActiveX control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.

Entry in Attributes Window	Represents
Properties...	Displays a dialog box for editing the properties provided with the selected ActiveX control. To enable editing a property, select it from the "Properties" list box. Only simple properties are displayed in this list box. Other properties (for example parameterized properties) can be configured using the ActiveX control’s property pages. See ActiveX Control Property Pages. The current value of the selected property is displayed in the "Value" field. If the ActiveX control does not allow reading of the current value, the field is captioned "Value (write-only)" and the value is not displayed. The "Value" field appears as a text box or a combo box, depending on the type of property. There are three ways to edit a property: 1. If "Value" appears as a text box, type in the value and use the "Apply" button to indicate that you have finished editing. 2. If "Value" appears as a combo box, you must pull down the combo box and select an entry. 3. If an additional dialog box is provided to select a value for the property, the "Select..." button is enabled. Choose the "Select..." button. In the dialog box that appears, select a value. Return to the "Properties" dialog box. To confirm, choose the "Apply" button. To reset a property to its initial value, use the "Reset" button. **Note:** The initial value is not displayed as long as the "Properties" dialog box is still open. It is valid, though, after the dialog box has been closed. If you can edit the value of the property directly, the default value is displayed in the "Value" field. To select a value other than the default value, overwrite it. For help on the selected property, click the "Help" button. (The help file for the ActiveX control must have been installed). To confirm the property settings, choose "Close".
About...	Dialog box with information on the ActiveX control.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

ActiveX Control Property Pages

▶ **Accessible Using**

1. if selected: "Control > Property Pages..."; or
2. "Property Pages..." from the context menu.

Property pages are available with most ActiveX controls. They are used to configure the attributes of the ActiveX control in an individual way. After an ActiveX control in a dialog has been configured using its property pages, Natural stores the result of the configuration in binary form in the private resource file associated with the dialog. For more information on resources see the Programming Guide - Object Types - Resource.

Bitmap Control Attributes Window

▶ **Accessible Using**

1. Double-click on the bitmap control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the bitmap control (may be overwritten with another name).
Array...	"Array" dialog box for defining an array of bitmap controls.
Bitmap	BITMAP-FILE-NAME attribute value. If you pull down the selection box, you can choose from the existing set of .bmp files.
. . .	"Source" dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available bitmaps to be used.
DIL Text	DIL-TEXT attribute value (string).
. . .	"Source" dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
State:	
Visible	VISIBLE attribute value.
Draggable	DRAGGABLE attribute value. If you check this item, the end user may drag the bitmap control and drop it onto another bitmap control.
Enabled	ENABLED attribute value.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlpfile).
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).

Entry in Attributes Window	Represents
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Background Color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value. If ’default’ is specified, the color of the first (top-left) pixel in the bitmap determines the background color.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Rectangle:	The following four attributes decide the bitmap control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Style:	
Vertical Justification:	
Top / Center /	Mutually exclusive STYLE attribute values: align to the
Bottom	bottom, the vertical center, the top.
Horizontal Justification:	
Left / Center /	Mutually exclusive STYLE attribute values: align the bitmap
Right	to the left (of the rectangle), the horizontal center, the right.
Framed	STYLE attribute value: three-dimensional frame.
Scaled	STYLE attribute value: scale the bitmap to fit into the underlying bitmap control’s rectangle.
Transparent	STYLE attribute value: bitmap pixels in the background color do not change the state of the screen.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Canvas Control Attributes Window

Accessible Using

1. Double-click on the canvas control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	**Represents**
Name	Handle name of the canvas control (may be overwritten with another name).
Array...	"Array" dialog box for defining an array of canvas controls.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL Text	DIL-TEXT attribute value (string).
. . .	"Source" dialog box for determining sources of DIL-TEXT attribute values.
Help ID	HELP-ID attribute value. You must use the help topic's .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Style:	
Frame	STYLE attribute value: creates a frame around the canvas control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Rectangle:	The following four attributes decide the canvas control's x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground Color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Control Box Control Attributes Window

Accessible Using

1. Double-click on the control box control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the control box control.
State	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Style:	
Framed	STYLE attribute value; creates a simple frame around the control box control.
Lowered	STYLE attribute value; creates a 3-D border with a sunken appearance.
Exclusive	STYLE attribute value; marks the control box as exclusive. Amongst any set of sibling controls (i.e., controls with the same parent), only one control box marked as exclusive can be visible at any one time. This applies both in the Dialog Editor and at run-time.
Transparent	STYLE attribute value; creates a transparent control box. Allows the control box itself to be invisible without making the child controls it contains invisible.
Size to parent	STYLE attribute value; control boxes with this style are resized to fill the entire client area of their parent whenever the parent control is resized, or when this style is initially set in the Dialog Editor.
Rectangle:	The following four attributes decide the control box control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Invokes dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Dialog Attributes Window

▶ **Accessible Using**

1. Double-click on the dialog background; or
2. "Dialog > Attributes"; or by selecting ’Attributes...’ from the dialog’s context menu or
3. ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the dialog window (may be overwritten with another name).
Type	TYPE attribute value. Allows you to decide whether the dialog provides a Multiple Document Interface (MDI frame or MDI child) or not (standard).
String	STRING attribute value (title of the "Dialog" window).
. . .	"Source" dialog box for determining sources of STRING attribute values. For more information on the "Source" dialog box, see Source.
Font	Value of the FONT-STRING attribute. Decides the font for all dialog elements in this dialog except for the system-supplied window decorations and the dialog elements for which a font has been chosen explicitly.
. . .	"Font" dialog box for determining sources of STRING attribute values. For more information on the "Font" dialog box, see Source.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the dialog itself.
Icon	BITMAP-FILE-NAME attribute value. Also provides a list of all available icons to be used.
Help file	HELP-FILE-NAME attribute value. Decides a dialog’s help file name (without extension).
Default button	DEFAULT-BUTTON attribute value: type in or select the handle name of the push-button control for which you want to assign this attribute.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Docking	DOCKING attribue value. Determines the sides of the dialog (if any) on which tool bars are allowed to dock.
Background Color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value. Choose a predefined color.
. . .	"Custom" dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Style:	
Modeless (Popup) /Modal /Dialog box	Mutually exclusive values for the STYLE attribute.

Entry in Attributes Window	Represents
Centered position	STYLE attribute value. Dialog will be centered on screen.
Default position	STYLE attribute value. If set, the initial position (but not size) of the dialog is determined by the windowing system. The setting will be ignored if "Dialog box" is set. This option is especially useful for MDI child dialogs.
Default rectangle	Value of the STYLE attribute. If set, the initial position and size of the dialog are decided by the windowing system. The setting will be ignored if "Dialog box" is set.
Control clipping	Value of the STYLE attribute. If set, dialog elements are not allowed to overpaint other dialog elements with the same parent which occur later in the control sequence.
3-D client window	STYLE attribute value. If set, the dialog interior is drawn with a sunken 3-D appearance.
State:	
Visible	VISIBLE attribute value. If you check this entry, the dialog is visible.
Enabled	ENABLED attribute value. If you check this entry, the end user may interact with the dialog.
Maximized	MAXIMIZED attribute value. If you check this entry, the dialog is maximized to fill the entire screen.
Minimized	MINIMIZED attribute value. If you check this entry, the dialog is minimized to icon size. The end user then will have to double-click on the icon to restore the dialog to its default size.
Save layout	If checked, the tool and status bar control layout will be automatically saved and restored on a per-user basis between sessions at run-time. This option is only available if the dialog contains at least one tool bar or status bar control.
Popup help	POPUP-HELP attribute value. Help for this dialog or any of its controls will be displayed in a popup window.
Auto-adjust	AUTOADJUST attribute value. If you check this entry, the dialog will be scaled at run time according to the current system font size (i.e. "large fonts"/"small fonts" setting).
Event queueing	EVENT-QUEUEING attribute value. If you check this entry, messages for this dialog are queued instead of being processed immediately.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value. Choose a predefined color.
. . .	"Custom" dialog box for editing
	BACKGROUND-COLOUR-VALUE attribute value.
Components:	
Menu bar	MENU-HANDLE attribute value: if checked, the dialog editor will assign the handle value specified in the menu bar attributes window.
Toolbar	HAS-TOOLBAR attribute value: if checked, the dialog editor will assign the handle value specified in the toolbar attributes window and set HAS-TOOLBAR to TRUE.

Entry in Attributes Window	Represents
Status bar	HAS-STATUS-BAR attribute value. If you check this entry, the dialog has a status bar.
Dynamic info line	HAS-DIL attribute value. If you check this entry, the dialog has a dynamic information line.
System button	HAS-SYSTEM-BUTTON attribute value. If you check this entry, the dialog has a system button.
Size modifiable	SIZE-MODIFIABLE attribute value. If you check this entry, the dialog’s size may be modified.
Maximizable	MAXIMIZABLE attributes value. If you check this entry, the dialog may be maximized.
Minimizable	MINIMIZABLE attribute value. If you check this entry, the dialog may be minimized.
Horizontal scroll bar	HORIZ-SCROLLABLE attribute value. If you check this entry, the dialog has a horizontal scroll bar.
Vertical scroll bar	VERT-SCROLLABLE attribute value. If you check this entry, the dialog has a vertical scroll bar.
Help button	HAS-HELP-BUTTON attribute value. If you check this entry, the dialog title bar contains a help button. Note: Windows does not display the help button if minimize and maximize buttons are present.
Rectangle:	The following four attributes decide the dialog’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Subroutines	Dialog box for editing subroutines.
Help	Provides online help on the attributes window.

Dialog Context Menus Window

Accessible Using

1. "Dialog > Context Menus"; or
2. CTRL+ALT+X.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Available Context Menus	Shows the context menus currently defined for this dialog. One or more context menus can be selected from this list.
New	Creates a new, empty, context menu with a default name. The new context menu is inserted into the list immediately after the selected context menu(s).
Cut	Cuts the selected context menu(s) to the clipboard.
Paste	Pastes context menu(s) from the clipboard, which are inserted into the list immediately after the selected context menu(s).
Selected Context Menu	Shows information relating to the currently selected context menu. If multiple context menus are selected, this section is disabled.
Name	Displays the name of the currently selected context menu, which can be modified here or in the Menu Editor Window itself.
Enabled	The initial ENABLED attribute state for the context menu. A disabled context menu is suppressed at run-time.
Edit	Invokes the menu editor for the selected context menu, where the menu items themselves can be defined.
Events	Invokes the event handler window for editing the events for the context menu itself. (The events for the menu items and any submenus are accessed using the Menu Editor Window).
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Edit Area Control Attributes Window

Accessible Using

1. Double-click on the edit area control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the edit area control (may be overwritten with another name).
Array...	"Array" dialog box for defining an array of edit area controls.
String	STRING attribute value.
. . .	"Source" dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL Text	DIL-TEXT attribute value (string).
. . .	"Source" dialog box for determining sources of DIL-TEXT attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Modifiable	MODIFIABLE attribute value. If this entry is checked, the end user may edit the text.
Horizontal scroll bar	HORIZ-SCROLLABLE attribute value.
Vertical scroll bar	VERT-SCROLLABLE attribute value.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Length	LENGTH attribute value. Specifies the maximum number of characters which can be entered into the edit area control. **Note:** Each line break consumes two characters (carriage return / line feed). This applies regardless of whether the line break was explicitly entered by the user or implicitly inserted due to word wrapping.
Style:	
Framed	STYLE attribute value: creates a frame around the edit area control.

Entry in Attributes Window	**Represents**
Wordwrapped	STYLE attribute value: when text exceeds the width of the edit area control, it is automatically wrapped to the next line. **Note:** When you set the STYLE attribute value to "WORDWRAP", you cannot set the HORIZ-SCROLLABLE attribute value to "TRUE" and vice versa.
Autoscroll	STYLE attribute value: Text is vertically scrollable and is automatically scrolled upwards when the ENTER key is pressed on the last displayed line. **Note:** This option only has an effect if the edit area control does not have a vertical scroll bar. Otherwise, the text is implicitly autoscrollable.
Rectangle:	The following four attributes decide the edit area control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Group Frame Control Attributes Window

Accessible Using

1. Double-click on the group frame control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the group frame control (may be overwritten with another name).
Array...	Dialog box for defining an array of group frame controls.
String	STRING attribute value.
. . .	"Source" dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
Style:	
Container	STYLE attribute value. If checked, all existing controls within the group frame, and any controls created within it, become children of the group frame.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	"Custom" dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	"Custom" dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Rectangle:	The following four attributes decide the group frame control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Input Field Control Attributes Window

Accessible Using

1. Double-click on the input field control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu.
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the input field control (may be overwritten with another name).
Array...	Dialog box for defining an array of input field controls.
String	STRING attribute value.
. . .	"Source" dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Modifiable	MODIFIABLE attribute value. If this entry is checked, the end user may edit the text.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Edit mask	EDIT-MASK attribute value (only enabled if STRING source is a linked variable).
Length	LENGTH attribute value.
Left / Center / Right	Mutually exclusive STYLE attribute values: align input to the left, the center, the right. **Note:** When you create an input field control, and you assign it a STYLE value of "Center" or "Right", the input field control must be higher than the font. Otherwise, the STRING will not be displayed.
Mandatory	STYLE attribute value: input is mandatory.
Upper case	STYLE attribute value: input will be converted to UPPERCASE letters.
Nondisplay	STYLE attribute value: input is displayed as a series of asterisks (for example, for passwords).

Entry in Attributes Window	Represents
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Rectangle:	The following four attributes decide the input field control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

List Box Control Attributes Window

Accessible Using

1. Double-click on the list box control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the list box control (may be overwritten with another name).
Items	Input field where you can specify the number of list box items in the list box control. When you enter a number here, the dialog editor generates the corresponding list box items and the "Source" dialog box becomes enabled.
. . .	Dialog box for determining sources of the list box items’ STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Multiple selection	MULTI-SELECTION attribute value. If you check this entry, the end user may select several list box items at a time.
Sorted	SORTED attribute value. If you check this entry, the items are sorted and you cannot modify them.
Autoselect	AUTOSELECT attribute value. If you check this entry, Natural automatically updates the selection in response to a right mouse button click before displaying a context menu.
Style:	
3-D Border	STYLE attribute value: list box has sunken appearance.
Integral height	STYLE attribute value: partial rows are not displayed.

Entry in Attributes Window	Represents
Rectangle:	The following four attributes decide the list box control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	"Custom" dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	"Custom" dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Menu Editor Window

Accessible Using

1. First check the "Menu Bar" field in the "Dialog Attributes" window, then double-click on the dummy menu bar in the dialog; or
2. "Dialog > Menu Bar" or by selecting ’Menu Bar...’ from the menu bar’s context menu or
3. CTRL+ALT+M.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Menu:	
Name	Handle name of the menu (may be overwritten with another name).
Submenus	Lists the menu’s handle name and all of the menu items of MENU-ITEM-TYPE "Submenu" defined so far. This list is indented, that is, the menu structure becomes visible. If you select an entry, its *children* menu items or submenu controls appear in the "Selected Submenu" group frame.
Selected submenu:	Displays the STRING attribute values of the menu items or submenu controls which have been created as child of the "Submenus". You can edit the attributes of the currently "Selected Submenu" in the "Selected Menu Item" group frame. The entries marked ">" are submenus. (You can also select several menu items for cutting and pasting.)
Cool menu	Enables the display of bitmaps within a menu. If not checked, no menu bitmaps will be displayed, even if the menu items themselves have a bitmap assigned to them.
Image width	The width of images to be displayed alongside the menu items. Menu item bitmaps with a different width will be scaled, or truncated or extended (in the background color) to fit, depending on the value of the menu item’s ’scaled’ attribute.
Image height	The height of images to be displayed alongside the menu items. Menu item bitmaps with a different height will be scaled, or truncated or extended (in the background color) to fit, depending on the value of the menu item’s ’scaled’ attribute. The specified image height may determine the menu item height if larger than the standard menu item height will allow.
Menu items:	Displays the STRING attribute values of the menu items or submenu controls which have been created as child of the "Submenus". You can edit the attributes of the currently "Selected Submenu" in the "Selected Menu Item" group frame. The entries marked ">" are submenus. (You can also select several menu items for cutting and pasting.)

Entry in Attributes Window	**Represents**
<< Parent menu **Submenu >>**	When you are creating a menu hierarchy, these two push buttons enable you to navigate to the next higher level (<< Parent Menu) or the next lower level (Submenu >>) of the existing branches.
Selected menu item:	Displays the attribute values of the selected submenu for editing. For editing, it is necessary that one menu item be selected.
Name	Handle name of the menu item or submenu control (may be overwritten with another name starting with the # sign).
Type	MENU-ITEM-TYPE attribute value for the selected menu item. If the type is "Submenu" or "Window submenu", this item is automatically changed into a submenu control.
Same as	SAME-AS attribute value (only available for MENU-ITEM-TYPE "Normal"); the selection box displays the signals available. If this field is filled, the fields for the attributes which are inherited from the referenced signal are disabled, and can only be re-enabled if the link is broken again by deleting the "Same as" field contents.
OLE	MENU-ITEM-OLE attribute value. If a dialog has a menu bar and an OLE container control is being edited in-place, this attribute decides whether a top-level menu item or a submenu control is not an OLE menu item, or whether it is an item that represents the OLE Container or File or Window group.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Bitmap	BITMAP-FILE-NAME attribute value.
. . .	Dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available bitmaps to be used.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
Background Color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value to be used for display of the menu item’s bitmap (if any). If ’default’ is specified, the color of the first (top-left) pixel in the bitmap determines the background color.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
State:	
Enabled	ENABLED attribute value.
Shared	SHARED attribute value. CLICK events for this menu item will be forwarded to the active MDI child dialog (if any). This attribute is ignored for non-MDI dialogs.
Checked	CHECKED attribute value (not applicable to submenu controls).

Entry in Attributes Window	**Represents**
Style:	
Scaled	STYLE attribute value: scale the menu item’s bitmap to fit the image height and width specified for the submenu.
Transparent	STYLE attribute value: menu item bitmap pixels in the background color do not change the state of the screen.
Events	Dialog box for editing event handlers; may only be used with the appropriate "Type" field.
New	Creates a new submenu control or menu item. If you change the type in the "Type" field of the "Selected Menu Item" group frame, it creates a menu item with a corresponding MENU-ITEM-TYPE attribute value. Within a submenu, it creates a menu item.
Cut	Cuts the selected menu item(s) and copies it (them) to the clipboard.
Copy	Copies the selected menu item(s) to the clipboard.
Paste	Pastes menu item(s) from the clipboard. **Note:** The "New" and "Paste" entries insert menu items behind the currently selected item, or, if no items are selected, at the top of the list. You deselect items by holding down CTRL while clicking on the selected items.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

OLE Container Control Attributes Window

Accessible Using

1. Double-click on the OLE container control; or
2. if selected: "Control > Attributes"; or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the OLE container control (may be overwritten with another name).
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Object Information:	
	In this group box, you decide the OLE object’s type whose name is then displayed.
Type	Decides whether the OLE container control contains an OLE server, a new OLE object, an existing OLE object, or none of all. For more information on these three types, see Selecting an OLE Server or Document. **Note:** This is not a value of the TYPE attribute.
. . .	Dialog box for selecting a particular OLEserver, a new OLE object, or an existing Natural embedded OLE object. EMBEDDED-OBJECT, SERVER-OBJECT and SERVER-PROGID attribute values.
Name	Displays the name of the selected item. You cannot edit this entry.
Framed	STYLE attribute value: draw a frame around the OLE container control.
Zoom (%)	ZOOM-FACTOR attribute value: magnify or reduce the default representation of an OLE server application that has become visible in an OLE container control.
Status:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Modifiable	MODIFIABLE attribute value. If this entry is checked, the end user may modify the OLE object in-place.

Entry in Attributes Window	Represents
Rectangle:	The following four attributes decide the OLE container control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK & Start Server	Save settings, start the OLE server and exit the window.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Selecting an OLE Server or Document

If you select an OLE server or document, the three options "OLE server", "OLE object" and "Existing OLE object" imply a number of restrictions when using Natural and when using the server application.

Differences Between OLE Server, New OLE Object and Existing OLE Object

Before you select an entry, decide if this is what you need.

Type	Characteristics
OLE Server	Creates an OLE object in its native form. Either server with no content ("Create New") or server with existing file as content ("Create from File").
New OLE Object	Creates a new OLE embedded object to be stored within the Natural environment (default file extension ".neo"). Only "Create New" allowed.
Existing OLE Object	Creates an existing OLE embedded object that has been stored within the Natural environment (default file extension ".neo"). Only "Create from File" allowed.

OLE Server

If you have selected the "OLE server" entry

1. Select the "..." button to the right of the drop-down combo box.
   The "Select OLE Server or Document" dialog box appears. Here you have two options:
   - The "Create New" radio button enables you to select a server application to be started when the end user activates the OLE container control at runtime. The server application is started as such, with no file loaded into it.
   - The "Create from File" radio button enables you to insert the contents of a file as an OLE object. You can browse for the file. When the end user activates the OLE container control at runtime, the application used to create the file is started as a server application, with the content being the selected file.
2. Either select "Create New".
   Or select "Create from File".
   If you have selected "Create New", proceed with 3a.
   If you have selected "Create from File", proceed with 3b.
3. 3a. - From the "Object Type" list box, select an application, for example "Microsoft Word 6.0 Document".
   3b. - In the file text box, enter the path of the file you want to select.
   Or, if you are not sure where the file is, choose the "Browse" button to search in your environment.
4. Select the "Display as Icon" check box or not. This lets you decide whether you want to display your application or file as an application icon inside the OLE container control or whether you want your application or file to appear as a text string called <<*applicationname*>> or <<*pathname*>>. Both act as a placeholder for the server application.
   If you choose to display the application or file as an icon, you can customize the icon by selecting the "Change Icon..." button.
5. To save your settings and quit the dialog box, select OK.
   Or select "Cancel" to quit without saving.

To edit an OLE object inside an OLE container control at runtime

Prerequisite: you have selected "OLE server".

1. Click and hold down the right mouse button inside the OLE container control’s rectangle.
   The pop-up menu specific to your server application appears, saying for example:
   Edit *object-type*Object; or
   object-type Object with the submenus "Edit" and "Open".
   "Open" activates the server application in a separate window and enables you to edit and save the object. You can then quit the server application and return to Natural. "Edit" lets you activate the server application inside your Natural dialog.
2. Make your selection in the pop-up menu.
3. Edit (and save) your object using the menu entries provided by the OLE server application.

To quit the OLE server application at runtime

If you have chosen "Edit":

- Click outside the OLE container control’s rectangle.
  The OLE server application is deactivated in the Natural dialog, but the object is still displayed inside the OLE container control.

If you have chosen "Open":

- From the OLE server application’s menu, select "File", then "Close and Return to *container-application-name*".
  The object is unloaded from the OLE server application in the separate window, but the object is still

displayed inside the OLE container control.

New OLE Object

If you have selected the "New OLE object" entry, do the following:

1. Select the "..." button to the right of the drop-down combo box.
   The "Select OLE Server or Document" dialog box appears. Important: Here you may only select "Create New", even though the other option is not disabled.
2. Select "Create New".
3. From the "Object Type" list box, select an application, for example "Microsoft Word 6.0 Document".
4. Select the "Display as Icon" check box or not. This lets you decide whether you want to display your object as an application icon inside the OLE container control or whether you want your object to appear as a text string called <<*applicationname*>>.
   If you choose to display the application or file as an icon, you can customize the icon by selecting the "Change Icon..." button.
5. To save your settings and quit the dialog box, select OK.
   Or select "Cancel" to quit without saving.
   You have returned to the attributes window. Note that your server's name now appears in the "Name" text box, prepended by an "@". This is the current value of the "SERVER-PROGID" attribute. The OK button is disabled. Instead, the "OK & Start Server" button is enabled.
6. Ensure you have made all choices in the attributes window.
7. Choose "OK & Start Server".
   Your attributes window settings are saved and the server application is started.
8. Create your OLE object.
9. Quit the server application. The server application usually provides the menu entries "File", then "Close and Return to *container-application-name*".
   A file list box called "Save As" appears.
10. Save the file as a Natural embedded object with the default file extension ".neo".

If your end user has edited your new OLE object at runtime

1. To edit it in-place, the server application provides additional entries to the Natural application's menu bar.
2. To quit the server application, the server application usually provides the menu entries "File", then "Close and Return to *container-application-name*".
   Note:
   Depending on the server application, you might have to set the focus back to Natural as the server applications usually remain active.
   A file list box called "Save As" appears.
3. The end user must save the file as a Natural embedded object with the default file extension ".neo".

Existing OLE Object

If you have selected the "Existing OLE object" entry:

1. Select the "..." button to the right of the drop-down combo box.
   The "Select existing Natural Embedded Object" dialog box appears. It displays all Natural embedded objects with the default file extension ".neo" in the default directory.
2. Select a file.
   Both the OK and the "OK & Start Server" buttons are enabled. You now have two options:
   - If you quit the attributes window by selecting OK, the embedded object will be shown in the container, but cannot be modified (read-only).
   - If you quit the attributes window by selecting "OK & Start Server", the corresponding server application is started and the chosen object can be modified (read-write).
3. Choose "OK & Start Server".

Or choose OK.
Your attributes window settings are saved and the server application is started.

4. Modify your OLE object (if you have chosen "OK & Start Server" and the object is read-write).
   Or look at your OLE object (if you have chosen OK and the object is read-only).
5. Quit the server application. The server application usually provides the menu entries "File", then "Close and Return to *container-application-name*".
   A file list box called "Save As" appears.
6. If you confirm the default, the file is automatically saved as a Natural embedded object with the default file extension ".neo".

Push Button Control Attributes Window

► Accessible Using

1. Double-click on the push-button control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the push-button control (may be overwritten with another name).
Array...	Dialog box for defining an array of push-button controls.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Style:	
OK Button	STYLE attribute value: if the end user presses ENTER, this button is pushed.
Cancel Button	STYLE attribute value: if the end user presses ESC, this button is pushed.
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
Rectangle:	The following four attributes decide the push button control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Radio Button Control Attributes Window

Accessible Using

1. Double-click on the radio-button control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected:ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the radio-button control (may be overwritten with another name).
Array...	Dialog box for defining an array of radio-button controls.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Checked	CHECKED attribute value.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Group ID	GROUP-ID attribute value (means this radio-button control belongs to the group of radio buttons with this ID).
Rectangle:	The following four attributes decide the radio-button control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground color:	

Entry in Attributes Window	**Represents**
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Scrollbar Control Attributes Window

Accessible Using

1. Double-click on the scroll-bar control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected:ENTER

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the scroll-bar control (may be overwritten with another name starting with the # sign).
Array...	Dialog box for defining an array of scroll-bar controls.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Values:	
Minimum	MIN attribute value (minimum numerical value on the scale).
Maximum	MAX attribute value (maximum numerical value on the scale).
Line	LINE attribute value (number of logical units by which the slider moves if the end user presses the up and down arrow buttons).
Page	PAGE attribute value (number of logical units by which the slider moves if the end user clicks on the scroll-bar control’s shaft).
Slider	SLIDER attribute value (position of the slider in between the MIN and MAX values).
Rectangle:	The following four attributes decide the scroll-bar control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.

Entry in Attributes Window	Represents
Horizontal / Vertical	Mutually exclusive STYLE attribute values: slider will scroll horizontally or vertically. **Note:** When you edit the STYLE attribute value in the scroll-bar control attributes window, setting "h" instead of "v" and vice versa, the RECTANGLE-H and RECTANGLE-W attribute values are exchanged. The dialog editor thus ensures that the scroll-bar control will not provide for vertical scrolling in a horizontal shape and vice versa.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Selection Box Control Attributes Window

Accessible Using

1. Double-click on the selection box control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the selection box control (may be overwritten with another name).
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Items	Input field where you can specify the number of selection box items in the selection box control. When you enter a number here, the dialog editor generates the corresponding selection box items and the corresponding "Source" dialog box becomes enabled.
. . .	Dialog box for determining sources of the selection box items’ STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Modifiable	MODIFIABLE attribute value.
Sorted	SORTED attribute value. If you check this entry, the items are sorted and you cannot modify them.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Edit mask	EDIT-MASK attribute value.
Length	LENGTH attribute value.

Entry in Attributes Window	**Represents**
Rectangle:	The following four attributes decide the selection box control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Mandatory	STYLE attribute value: input in the selection box control’s input field is mandatory.
Box dropped down	STYLE attribute value: the box stays dropped down all the time.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Signal Attributes Window

▶ Accessible Using

1. "Dialog> Signals"; or
2. CTRL+ALT+N.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Signals:	Displays the handle name of the signals already created. If you click on a signal in the list, its attributes are displayed for editing. You can also select several signals for cutting and pasting.
New	Creates a new signal.
Cut	Cuts the selected signal and copies it to the clipboard. You can also cut and paste several signals at once.
Copy	Copies the selected signal(s) to the clipboard.
Paste	Pastes a signal from the clipboard. **Note:** The "New" and "Paste" entries insert signals behind the currently selected signal, or, if no signals are selected, at the top of the list. You deselect items by holding down CTRL while clicking on the selected items.
Selected signal:	In this group frame, you assign attribute values to the signals selected in the "Signals" list box on the left.
Name	Handle name of the signal (may be overwritten with another name).
Type	MENU-ITEM-TYPE attribute value for the selected signal.
Bitmap	BITMAP-FILE-NAME attribute value.
. . .	Dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available bitmaps to be used.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Tooltip	TOOLTIP attribute value.
...	Dialog box for determining sources of TOOLTIP attribute values.
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
Background Color:	

Entry in Attributes Window	Represents
Selection box	BACKGROUND-COLOUR-NAME attribute value to be used for display of the signal’s bitmap (if any). If ’default’ is specified, the color of the first (top-left) pixel in the bitmap determines the background color.
...	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Checked	CHECKED attribute value.
Shared	SHARED attribute value. If checked, CLICK events for this signal will be forwarded to the active MDI child dialog (if any). This attribute is ignored for non-MDI dialogs.
Events	Dialog box for editing event handlers; may only be used with the appropriate "Type" entry.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Status Bar Control Attributes Window

Accessible Using

1. Double-click on the status bar control; or
2. if selected: "Control>Attributes " or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the status bar control (may be overwritten with another name)
Attributes...	Subordinate window for editing the status bar control’s attribute values. For more information, see Status Bar Control Attributes Subwindow. (Normally, all attributes of a dialog element can be edited in the attributes window. Instead, the attributes of each pane in the status bar control can be edited here. For reasons of space, the status bar control’s attributes are edited in a separate subwindow).
Status-bar panes:	Displays the handle name of the panes already created. If you click on a pane in this list, its attributes are displayed for editing. You can also select several panes for cutting and pasting.
New	Creates a new pane.
Cut	Cuts the selected pane and copies it to the clipboard. You can also cut and paste several panes at once.
Paste	Pastes a pane from the clipboard. **Note:** The "New" and "Paste" entries insert panes behind the currently selected item, or, if no items are selected, at the top of the list. You deselect items by holding down CTRL while clicking on the selected items.
Selected status bar pane:	In this group frame, you assign attribute values to the panes selected in the "Status bar panes" list box on the left.
Name	Handle name of the pane (may be overwritten with another name).
Width	ITEM-W attribute value. Specifies the width of the pane in pixels. **Note:** If 0, the pane does not have a fixed width, but is instead automatically sized to fill the space available ("stretchy pane").
String	STRING attribute value. Specifies the initial pane text.
. . .	Dialog box for determining sources of STRING attribute values.

Entry in Attributes Window	Represents
Icon	BITMAP-FILE-NAME attribute value. Species the icon (if any) to be displayed alongside the pane text. **Note:** Natural attempts to extract the small (16x16 pixel) icon (if any) from the specified icon file. If only a large (32x32 pixel) icon is present, Windows will automatically synthesize a small icon from it, which may lead to undesirable scaling effects.
. . .	Dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available icons to be used.
Tooltip	TOOLTIP attribute value.
...	Dialog box for determining sources of TOOLTIP attribute values.
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Shared	SHARED attribute value. If checked, CLICK events for this pane will be forwarded to the active MDI child dialog (if any). This attribute is ignored for non-MDI dialogs.
Style:	
Centered	STYLE attribute value. If set, text will be horizontally centered within the pane.
Hide disabled	STYLE attribute value. If set, the pane text and icon (if any) will be hidden (instead of being grayed out) when the pane is disabled.
Raised	STYLE attribute value. If set, the pane appears to "pop out".
No borders	STYLE attribute value. If set, the pane borders are not drawn. This style is typically applied to stretchy panes.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Status Bar Control Attributes Subwindow

Accessible Using

1. Click on the "Attributes..." button in the status bar control attributes window

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attrib. Subwindow	Represents
Control ID	CLIENT-KEY attribute value (used in this context for associating a user-defined ID).
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Location	LOCATION attribute value. Determines the side of the dialog on which the status bar control is initially positioned.
Internal Metrics:	
Minimum height	ITEM-H attribute value. Specifies the *minimum* height of the status bar panes (in pixels). This is particularly useful for status bar controls which display icons. By default, the minimum height of a status bar control depends on the font used to draw the text.
Margin-X	MARGIN-X attribute value. Specifies the margin (in pixels) to the left and right of the status bar panes.
Margin-Y	MARGIN-Y attribute value. Specifies the margin (in pixels) above and below the status bar panes.
Borders:	
Top	STYLE attribute value. Species whether a border should be displayed at the top of the control.
Bottom	STYLE attribute value. Species whether a border should be displayed at the bottom of the control.
3-D	STYLE attribute value. If set, the status bar control borders (if any) are drawn with a 3-D appearance.
Style:	
Gripper	STYLE attribute value. Determines whether a sizing gripper is displayed within the status bar control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Tooltips	HAS-TOOLTIP attribute value. If not set, display of the tool tip text (if any) for the status bar panes will be suppressed.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Table Attributes Window

Accessible Using

1. Double-click on the table; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Table:	
Name	Handle name of the table (may be overwritten with another name)
Attributes...	Subordinate window for editing the table’s attribute values. For more information, see Table Attributes Subwindow. (Normally, all attributes of a dialog element can be edited in the attributes window. Instead, the attributes of each column specification control in the table can be edited here. For reasons of space, the table’s attributes are edited in a separate subwindow).
Columns:	Displays the handle name, the COLUMN-TYPE and the STRING attribute values of the column specification controls already created. If you click on a column specification control, its attributes are displayed for editing. You can also select several column specifications for cutting and pasting.
Selected Column Specification:	In this group frame, you assign attribute values to the column specification controls selected in the "Columns" list box on the left.
Name	Handle name of the column specification control (may be overwritten with another name).
Type	COLUMN-TYPE attribute value for the selected column specification control. If the type is "Selection Box", the "Items" entry is enabled and enables you to define the number of selection box items and the source of their values.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Items	If the "Type" entry is set to "Selection Box", this entry is enabled and allows you to enter the number of selection box items.
. . .	Dialog box for determining sources of selection box item values.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
Width	RECTANGLE-W attribute value.
Length	LENGTH attribute value.

Entry in Attributes Window	Represents
<u>**State:**</u>	
Modifiable	MODIFIABLE attribute value.
New	Creates a new column specification control. If you change the type in the "Type" field of the "Selected Column Specification" group frame, it creates a column specification control with a corresponding COLUMN-TYPE attribute value.
Cut	Cuts the selected column specification control(s) and copies it (them) to the clipboard.
Paste	Pastes the selected column specification control(s) from the clipboard. **Note:** The "New" and "Paste" entries insert column specification controls behind the currently selected control, or, if no controls are selected, at the top of the list. You deselect controls by holding down CTRL while clicking on the selected controls.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Table Attributes Subwindow

▶ Accessible Using

- Click on the "Attributes..." button in the table attributes window.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attrib. Subwindow	**Represents**
Name	Handle name of the table (may be overwritten with another name).
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
Header font	Output field where the font currently selected for the table header is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
<u>Rectangle:</u>	The following four attributes decide the table’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Row count	ROW-COUNT attribute value.
Row height	ROW-HEIGHT attribute value.
Header height	HEADER-HEIGHT attribute value.
Width 1st col.	FIRST-COLUMN-WIDTH attribute value.
Frozen cols.	FROZEN-COLUMNS attribute value.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).
First visible col.	FIRST-VISIBLE-COLUMN attribute value.
First visible row	FIRST-VISIBLE-ROW attribute value.
Columns header	STYLE attribute value: buttons with field names are displayed at the top of each column.
Extendable	STYLE attribute value: end users can delete and insert rows using DEL and INS.
No lines	STYLE attribute value: the table control is displayed without the lines that normally separate the cells.
Resize columns	STYLE attribute value: end users may resize the columns horizontally.

Single cell selection	STYLE attribute value: if set, end users may only select single cells. If not set, end users may select ranges of cells.
Resize rows	STYLE attribute value: end users may resize the rows vertically.
Whole row selection	STYLE attribute value: selecting an individual cell sets the selection to the entire row.
Draggable columns	STYLE attribute value: if set, end users may drag the columns.
Integral height	STYLE attribute value: partial rows are not displayed.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Modifiable	MODIFIABLE attribute value.
Has first column	HAS-FIRST-COLUMN attribute value.
Horizontal scroll bar	HORIZ-SCROLLABLE attribute value.
Vertical scroll bar	VERT-SCROLLABLE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Text Constant Control Attributes Window

▶ Accessible Using

1. Double-click on the text constant control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the text constant control (may be overwritten with another name).
Array...	Dialog box for defining an array of text constant controls.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
Style:	
Left / Centered / Right	Mutually exclusive STYLE attribute values: align output to the left, the center, the right.
Framed	STYLE attribute value: draw a frame around the text constant control.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Rectangle:	The following four attributes decide the text constant control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Timer Attributes Window

▶ Accessible Using

1. "Dialog > Timers"; or
2. CTRL+ALT+I.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Timer:	Displays the handle names and the TIMER-INTERVAL attribute values of the timers already created. (You may create up to 16 timers per dialog).
Selected Timer:	In this group frame, you assign attribute values to the timer selected in the "Timers" list box on the left.
Name	Handle name of the timer (may be overwritten with another name starting with the # sign).
Interval	TIMER-INTERVAL attribute value.
Events	Dialog box for editing event handlers.
New	Creates a new timer.
Cut	Cuts the selected timer and copies it to the clipboard. (You can cut and paste one or several timers).
Paste	Pastes timer(s) from the clipboard. **Note:** The "New" and "Paste" entries insert timers behind the currently selected timer, or, if none is selected, at the top of the list. You deselect timers by holding down CTRL while clicking on the selected timers.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Toggle Button Control Attributes Window

Accessible Using

1. Double-click on the toggle-button control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the toggle-button control (may be overwritten with another name).
Array...	Dialog box for defining an array of toggle-button controls.
String	STRING attribute value.
. . .	Dialog box for determining sources of STRING attribute values.
Font	Output field where the font currently selected is displayed.
. . .	Dialog box for selecting fonts.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values.
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Help ID	HELP-ID attribute value. You must use the help topic’s .h file to map the numerical ID that you enter to the corresponding help topic ID (created by a markup in the .hlp file).
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value.
Checked	CHECKED attribute value.
Foreground color:	
Selection box	FOREGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing FOREGROUND-COLOUR-VALUE attribute value.
Background color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value.
. . .	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
Rectangle:	The following four attributes decide the toggle-button control’s x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Toolbar Attributes Window

Accessible Using

1. "Dialog > Toolbar"; or
2. first check the "Toolbar" entry in the dialog attributes window, then double-click on the dummy toolbar in the dialog; or select ’Toolbar...’ from the toolbar’s context menu.
3. CTRL+ALT+T.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the toolbar (may be overwritten with another name).
Position	TOOLBAR-POS attribute values.
Wrapped	STYLE attribute value: if set and there are more toolbar items than can be displayed on the top of the dialog, the toolbar wraps around to a new line. (The default: the toolbar can be scrolled with the two small arrow push buttons on the left of the toolbar.)
Margin-X	MARGIN-X attribute value (specifies which margin to the left, to the right and above the bitmaps is displayed in the toolbar area. This attribute only applies if TOOLBAR-POS is set to TB-LEFT or TB-RIGHT.
Margin-Y	MARGIN-Y attribute value (specifies which margin to the left, above and below the bitmaps is displayed in the toolbar area. This attribute only applies if TOOLBAR-POS is set to TB-TOP or TB-BOTTOM.
Item width	ITEM-W attribute value (specifies the width of all items in the toolbar).
Item height	ITEM-H attribute value (specifies the height of all items in the toolbar).
Toolbar	Displays the handle name and the BITMAP-FILE-NAME of the existing toolbar items.
Selected toolbar item:	In this group frame, you assign attribute values to the toolbar item selected in the "Toolbar items" group frame on the left.
Name	Handle name of the toolbar item (may be overwritten with another name).
Type	MENU-ITEM-TYPE attribute value for the selected toolbar item.
Same as	SAME-AS attribute value (not for MENU-ITEM-TYPE "Separator"); the selection box displays the menu items available.
Bitmap	BITMAP-FILE-NAME attribute value.
. . .	Dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available bitmaps to be used.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values (not for MENU-ITEM-TYPE "Separator").
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.

Entry in Attributes Window	Represents
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
Background Color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value to be used for display of the item’s bitmap (if any). If ’default’ is specified, the color of the first (top-left) pixel in the bitmap determines the background color.
...	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value (not for MENU-ITEM-TYPE "Separator").
Checked	CHECKED attribute value (not for MENU-ITEM-TYPE "Separator").
Shared	SHARED attribute value. CLICK events for this item will be forwarded to the active MDI child dialog (if any). This attribute is ignored for non-MDI dialogs.
Style:	
Scaled	STYLE attribute value: allows for stretched bitmaps to be displayed on the toolbar items.
Transparent	STYLE attribute value: bitmap pixels in the background color do not change the state of the screen.
Events	Dialog box for editing event handlers; may only be used with the approriate "Type" entry; may not be used if the toolbar item is associated with a menu item using the SAME-AS attribute.
New	Creates a new toolbar item.
Cut	Cuts a selected toolbar item and copies it to the clipboard. You can also cut and paste several toolbar items at once.
Paste	Pastes a toolbar item from the clipboard. **Note:** The "New" and "Paste" entries insert toolbar items behind the currently selected item, or, if no items are selected, at the top of the list. You deselect items by holding down CTRL while clicking on the selected items.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Tool Bar Control Attributes Window

Accessible Using

1. Double-click on the tool bar control; or
2. if selected: "Control > Attributes" or by selecting ’Attributes...’ from the control’s context menu or
3. if selected: ENTER.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Window	Represents
Name	Handle name of the tool bar control (may be overwritten with another name).
Attributes...	Subordinate window for editing the tool bar control's attribute values. For more information, see Tool Bar Control Attributes Subwindow. (Normally, all attributes of a dialog element can be edited in the attributes window. Instead, the attributes of each tool bar item in the tool bar control can be edited here. For reasons of space, the tool bar control's attributes are edited in a separate subwindow).
<u>Tool bar items:</u>	Displays the handle name and the BITMAP-FILE-NAME attribute values of the tool bar items already created. If you click on a tool bar item, its attributes are displayed for editing. You can also select several tool bar items for cutting and pasting.
New	Creates a new tool bar item.
Cut	Cuts a selected tool bar item and copies it to the clipboard. You can also cut and paste several tool bar items at once.
Paste	Pastes a tool bar item from the clipboard. **Note:** The "New" and "Paste" entries insert tool bar items behind the currently selected item, or, if no items are selected, at the top of the list. You deselect items by holding down CTRL while clicking on the selected items.
<u>Selected tool bar item:</u>	In this group frame, you assign attribute values to the tool bar items selected in the "Tool bar items" list box on the left.
Name	Handle name of the tool bar item (may be overwritten with another name).
Type	MENU-ITEM-TYPE attribute value for the selected tool bar item.
Width	RECTANGLE-W attribute value. This is only available for MENU-ITEM-TYPE "Separator" and specifies the separator width (0 = default separator width).
Same as	SAME-AS attribute value (only available for MENU-ITEM-TYPE "Normal"); the selection box displays the signals and menu items available.
Bitmap	BITMAP-FILE-NAME attribute value.
. . .	Dialog box for determining sources of BITMAP-FILE-NAME attribute values. Also provides a list of all available bitmaps to be used.
DIL text	DIL-TEXT attribute value (string).
. . .	Dialog box for determining sources of DIL-TEXT attribute values (not for MENU-ITEM-TYPE "Separator").
Accelerator	ACCELERATOR attribute value.
. . .	Dialog box for determining sources of ACCELERATOR attribute values.

Entry in Attributes Window	Represents
Tooltip	TOOLTIP attribute value.
. . .	Dialog box for determining sources of TOOLTIP attribute values.
Command ID	CLIENT-KEY attribute value (used in this context for associating a command ID).
Background Color:	
Selection box	BACKGROUND-COLOUR-NAME attribute value to be used for display of the item’s bitmap (if any). If ’default’ is specified, the color of the first (top-left) pixel in the bitmap determines the background color.
...	Dialog box for editing BACKGROUND-COLOUR-VALUE attribute value.
State:	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value (not for MENU-ITEM-TYPE "Separator").
Checked	CHECKED attribute value (not for MENU-ITEM-TYPE "Separator").
Shared	SHARED attribute value. CLICK events for this item will be forwarded to the active MDI child dialog (if any). This attribute is ignored for non-MDI dialogs.
Style:	
Scaled	STYLE attribute value: allows for stretched bitmaps to be displayed on the toolbar items.
Wrapped	STYLE attribute value: if set, tool bar item is started on a new row.
Transparent	STYLE attribute value: bitmap pixels in the background color do not change the state of the screen.
Events	Dialog box for editing event handlers; may only be used with the approriate "Type" entry; may not be used if the toolbar item is associated with a menu item using the SAME-AS attribute.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Tool Bar Control Attributes Subwindow

Accessible Using

1. · Click on the "Attributes..." button in the tool bar control attributes window.

Entries

Note:
For context-sensitive help on attribute entries, select the entry so it has the focus, and press F1.

Entry in Attributes Subwindow	Represents
String	STRING attribute value. This is the text displayed in the window caption when a dockable tool bar control is floated.
...	Dialog box for determining sources of STRING attribute values.
Control ID	CLIENT-KEY attribute value (used in this context for associating a user-defined ID).
Context Menu	CONTEXT-MENU attribute value. Specifies the context menu (if any) associated with the control.
Docking	DOCKING attribue value. Determines the sides of the dialog (if any) on which this tool bar is allowed to dock (if dockable). Note: The dialog itself must also support docking on the specified side(s).
Location	LOCATION attribute value. Determines the side of the dialog on which the tool bar control is initially positioned, or whether the tool bar control is floated in a separate window (if dockable).
Internal Metrics:	
Item width	ITEM-W attribute value (specifies the width of all items in the toolbar).
Item height	ITEM-H attribute value (specifies the height of all items in the toolbar).
Margin-X	MARGIN-X attribute value. Specifies the margin (in pixels) to the left and right of the tool bar items (for horizontal tool bars) or above and below the tool bar items (for vertical tool bars).
Margin-Y	MARGIN-Y attribute value. Specifies the margin (in pixels) above and below the tool bar items (for horizontal tool bars) or to the left and right of the tool bar items (for vertical tool bars).
Borders:	
Left	STYLE attribute value. Species whether a border should be displayed on the left side of the control. This option is not available for dockable tool bars.
Top	STYLE attribute value. Species whether a border should be displayed at the top of the control. This option is not available for dockable tool bars.
Right	STYLE attribute value. Species whether a border should be displayed on the right side of the control. This option is not available for dockable tool bars.
Bottom	STYLE attribute value. Species whether a border should be displayed at the bottom of the control. This option is not available for dockable tool bars.

Entry in Attributes Subwindow	Represents
<u>**Rectangle:**</u>	The following four attributes decide the tool bar control's x and y axis position, its height and its width on the screen. **X** - RECTANGLE-X attribute value. **Y** - RECTANGLE-Y attribute value. **W** - RECTANGLE-W attribute value. **H** - RECTANGLE-H attribute value. **Note:** the positions are relative to the dialog window.
<u>**Style:**</u>	
Gripper	STYLE attribute value. Determines whether a gripper bar is displayed within the tool bar control. Note: the gripper bar does not appear if the tool bar is floated.
Flat	STYLE attribute value. Indicates that the tool bar items should be displayed with a flat appearance.
Dynamic	STYLE attribute value. Indicates that the tool bar control can be resized when floated. Note: dynamic tool bars cannot contain any child controls.
3-D border	STYLE attribute value. If set, the tool bar control's border (if any) is drawn with a 3-D appearance.
<u>**State:**</u>	
Visible	VISIBLE attribute value.
Enabled	ENABLED attribute value (not for MENU-ITEM-TYPE "Separator").
Dockable	DRAGGABLE attribute value. If set, the tool bar control may be docked and/or floated in its own separate window.
Tooltips	HAS-TOOLTIP attribute value. If not set, display of the tool tip text (if any) for the tool bar items will be suppressed.
Flyby text	HAS-DIL attribute value. If not set, display of the DIL text (if any) for the tool bar items will be suppressed.
OK	Save settings and exit the window.
Cancel	Exit the window without saving the settings.
Help	Provides online help on the attributes window.

Dialog Boxes

- Array
- Data Area - Local, Parameter
- Data Area - Global
- Dialog Compile Error
- Events
- Import Data Field
- Font
- Source
- Subroutines

Array

Accessible Using

1. First open the attributes window of a dialog or dialog element by double-clicking on it or by pressing ENTER or by selecting ’Attributes...’ from the dialog or dialog element’s context menu.
2. Then click on the "Array..." push button.

Purpose

Define an array of dialog elements of the same type. This is especially useful for quickly creating a layout for end user input. An array of dialog elements will be treated as an entity by the dialog editor, that is, you can edit the entire array (move, resize, etc.). For example, you can create a column of evenly spaced input field controls plus a column of corresponding text constant controls.

Entries

Entry	Function
Dimensions	None means there will be no array, one means there will be a row or a column, two means there will be an array with both an x and a y axis.
Bounds	Dialog elements on the first and second axis from occurrence to occurrence.
Spacing	Number of pixels between occurrences aligned on the x and y axis.
Arrangement	Mutually exclusive options of how to arrange the dialog elements; the last axis is either the horizontal or the vertical one.

Data Area - Local, Parameter

Accessible Using

- "Dialog > Parameter Data Area/Local Data Area"; or
- CTRL+ALT+P/L, or (for LDA’s) by selecting ’Local Data Area...’ from the dialog’s context menu.

Purpose

Enter inline data definitions for a dialog. In a parameter data area, you must include all the parameters that you want to be passed on to the current dialog in an OPEN DIALOG or SEND EVENT statement. In a local data area, you must include all the user-defined variables or other variables that you want to use in an event handler code section or a subroutine of the current dialog. Note that the dialog editor automatically generates the data definitions for the dialog elements.

The "Using" button opens a dialog box that allows you to include existing inline data definitions.

Data Area - Global

Accessible Using

1. "Dialog > Global Data Area"; or
2. CTRL+ALT+G.

Purpose

Select an existing global data area from a list of available global data areas. To select, click on the entry in the list box. The data area is then displayed in the input field. To select, you can also enter the name of a global data area in the input field.

To create a new global data area, you use the data area editor.

Dialog Compile Error

Appears When

You check/run/stow a dialog, the compiler finds an error in the dialog's generated code, and you select "Edit" in the "Error" dialog box.

Purpose

Describes the error and lists the line of generated code together with the line number. If you press the OK push button, the section of the dialog appears where the compiler has located the error.

If you have saved your dialog sources in non-enhanced format and the enhanced listing option is enabled, any Natural error message will contain an incorrect line number. To ensure that you get the correct line number, disable the enhanced listing option. As under Natural for Windows and Unix/OpenVMS Version 4.1, dialog sources are always saved in enhanced format, this line number inconsistency does not exist.

To disable enhanced dialog list mode

- From the "Options" menu, choose "Enhanced dialog list mode".
  The menu item no longer has a check mark. This indicates the option is disabled.

Events

Accessible Using

1. Click on the "Events..." push button in an attributes window; or
2. "Dialog > Event Handlers" for dialog events; or
3. CTRL+ALT+E or SHIFT+ENTER for dialog events; or
4. "Control > Event Handlers" for a selected dialog element; or
5. CTRL+SHIFT+E or SHIFT+ENTER for a selected dialog element.

Purpose

Enter Natural event handler code for those events that are provided for the dialog or dialog element; also allows you to enter event handler code for user-defined SEND EVENTs.

Entries

Entry	Function
Event Name	This selection box lists the names of the system-provided events, such as the click event; it also lists the names of the user-written events that can be triggered by specifying SEND EVENT *user-written-event-name*. Please note that *user-written-event-names* are limited to 32 characters and that the option only applies to dialog events.
Editor	Invokes the program editor for the currently displayed event. Before using the program editor the dialog box must be closed using the ’OK’ push button.
Rename	(Only applies to dialog events). This push button opens a dialog box where you can rename a user-written event.
New	(Only applies to dialog events). This push button opens a dialog box where you can enter the name of a new, user-written event.
Clear	(Only applies to dialog events). This push button opens a message box where you can specify whether you want to delete the code of a system-provided event or the code and the name of a user-written event.
Use	This push button opens a dialog box where you can select a subprogram or a subroutine by choosing an item from a list of objects or by entering the object name in the input field. Depending on whether it is a subprogram or a subroutine, you get a display of whether the CALLNAT or the PERFORM statement will be used. After having selected the subprogram or subroutine, you leave the dialog box by choosing OK. The subprogram or subroutine will be used by your current event handler code section. At the position where you left the event handler section, you will find the CALLNAT or PERFORM statement with the name of the object.
Suppress	Suppresses an event for which a corresponding SUPPRESS-*eventname*-EVENT attribute exists. The event is also suppressed if you leave the event handler section empty.
Event Info...	(Only applies to ActiveX control events): Provides information on the parameters of each event.
(Edit area)	Here you enter your Natural code that you want to be triggered when the event occurs.
OK	Saves the code (and name) of the event handler section and exits the dialog box.
Cancel	Exits the dialog box without saving the settings.
Help	Provides online help.

Import Data Field

Accessible Using

- "Insert > Import > Input Field/Selection Box".

Purpose

Create an input field control or a selection box control based on a data field from another Natural object in another Natural library. The dialog element is created with a linked variable as the source of its STRING attribute value. You must declare this linked variable in a data area of the dialog.

Entries

Entry	Function
Library	Selection box where you can select the library containing the Natural object with the data field of your choice.
Type	Mutually exclusive options for Natural object types.
Object List	Once you have chosen a library and an object type, all objects with these criteria will be displayed here.
Data fields	Once you have chosen an object from the object list, all data fields defined in this object will be displayed.
Import	Once you have chosen one or several data field(s), you push this button and its content will be imported into your input field control or selection box control.
Cancel	Exits the dialog box without saving the settings.
Help	Provides online help.

Font

Accessible Using

1. First open the attributes window of a dialog or dialog element by double-clicking on it or by pressing ENTER.
2. Then click on the "..." push button next to the font selection box.

Purpose

Select a font type, such as "Times New Roman", a font face, such as "bold", and a font size, such as "10". After selecting your font, a sample is displayed. When you choose OK, a font control is generated and assigned to the FONT-HANDLE attribute of the dialog element you are currently editing.

Source

Accessible Using

1. First open the attributes window of a dialog or dialog element by double-clicking on it or by pressing ENTER.
2. Then click on the "..." push button to the left of an attribute entry.

Purpose

Define the source of attribute values, for example for the STRING attribute.

Entry	**Function**
Value	Current attribute value; the name of this entry varies depending on the attribute source.
Attribute Source:	
Constant	Text string.
Message file	Number of the string in the message file. If you have specified an array of dialog elements, the number of the first string appears here. The number of the string for each occurrence in the array is generated in ascending order from the first string number onwards.
Variable	When a dialog is opened with an OPEN DIALOG statement, the content of this variable will be assigned to the attribute. You can dynamically change the content of this variable in the before-open event handler.
	For more information, see Message Files and Variables as Sources of Attribute Values.
Linked variable	Only applicable to input field controls and selection box controls. The input of an end user will automatically be moved to this variable when the dialog element is left. When you have changed the content of a linked variable dynamically (during processing of an event handler section), you can use the PROCESS GUI statement action REFRESH-LINKS and the refreshed variable content will be displayed in the input field control or selection box control.
Array Values:	Attribute values if there is an array of dialog elements.
Individual values	Each occurrence in the array will have its individual attribute value.
Repeated single value	All occurrences in the array will have the same attribute value.
OK	Saves the settings and exits the dialog box.
Cancel	Exits the dialog box without saving the settings.
Help	Provides online help.

Subroutines

► Accessible Using

1. "Dialog > Inline subroutines"; or
2. CTRL+ALT+S.

Purpose

Enter standard sections of Natural code to be used in several event handler sections.

Entries

Entry	Function
Subroutine name	This selection box lists the names of the existing subroutines for the dialog.
Editor	Invokes the program editor for the currently displayed subroutine. Before you use the program editor, the dialog box must be closed using the ’OK’ push button.
Rename	This push button opens a dialog box where you can rename a subroutine.
New	This push button opens a dialog box where you can enter the name of a new subroutine. Subroutine names specified in the dialog editor are limited to 120 characters. The first 32 characters must be unique.
Delete	This push button opens a message box where you can specify whether you want to delete the code and the name of a subroutine.
Action:	Here you enter your Natural code in free form, that is, without having to specify the DEFINE SUBROUTINE and END-SUBROUTINE statements.
OK	Saves the code and name of the subroutine and exits the dialog box.
Cancel	Exits the dialog box without saving the settings.
Help	Provides online help.

Component Browser

The following topics are covered below:

- Introduction
- User Interface
- Application Development Support

Introduction

The Component Browser can be used to view ActiveX components which are available for developing NaturalX applications. It presents the information in a way that is especially useful to Natural application developers.

The Component Browser comprises the following features:

- Available ActiveX components and their dispatch and dual interfaces are listed.
- Data types are mapped to Natural formats.
- The external components’ help files are directly accessible.
- Natural programming examples are automatically generated.
- Many programming errors can be prevented.

User Interface

- Tree View
- Data View
- Interaction Tree View and Data View
- Menu

The Component Browser uses a split window. The left pane shows a tree view representing the available external components and the right pane shows information on a selected item.

Tree View

Groups

At startup the Component Browser’s tree view consists of four nodes that group the available external components.

All ActiveX Components

This group lists ActiveX Controls and Automation Objects.

ActiveX Controls

This group lists only the ActiveX Controls.

Automation Objects

This group lists the Automation Objects.

Interfaces

The last group lists all dual and dispatch interfaces that can be addressed in a Natural application. In this context, their relation to an ActiveX component is not taken into account.

Tree Nodes

In general, a node in the tree view represents either an ActiveX component or an interface. It provides textual information about the node and has a specific icon assigned that represents additional information.

The following table lists all available nodes with their icons and gives a short description:

Type	Icon	Description
Group	📁	Group node.
ActiveX component	⧉	ActiveX component.
Interface	(∴)	Interface of the current ActiveX component.
Default interface	∴	Default interface of the current ActiveX component.

Order

By default, the ActiveX component nodes are inserted in alphabetical order of their external names. If you wish to see them sorted according to their ProgID, select menu ’View > Show by ProgID’.

Interface nodes are always inserted in alphabetical order.

Data View

The data view uses a property sheet to display the specific information for a selected node. This sheet consists of four pages: **General, Properties, Events** and **Methods**.

General

Components and interface specific information such as name, Global Unique ID and help file name. It is always the active page if a new node is selected.

Properties

Specific information about the properties offered by an interface. These are, for example, the property’s name and type.

Events

Specific information on a component’s event interface. These are, for example, the event’s name and parameters.

Methods

Specific information about the methods and GET properties and PUT properties offered by an interface. Displayed are, for example, the method’s name, return type and parameters.

These pages are discussed in more detail in the context of interaction between tree and data view.

Interaction Tree View and Data View

The contents of the ’data view’ depend on the object selected in the tree view.

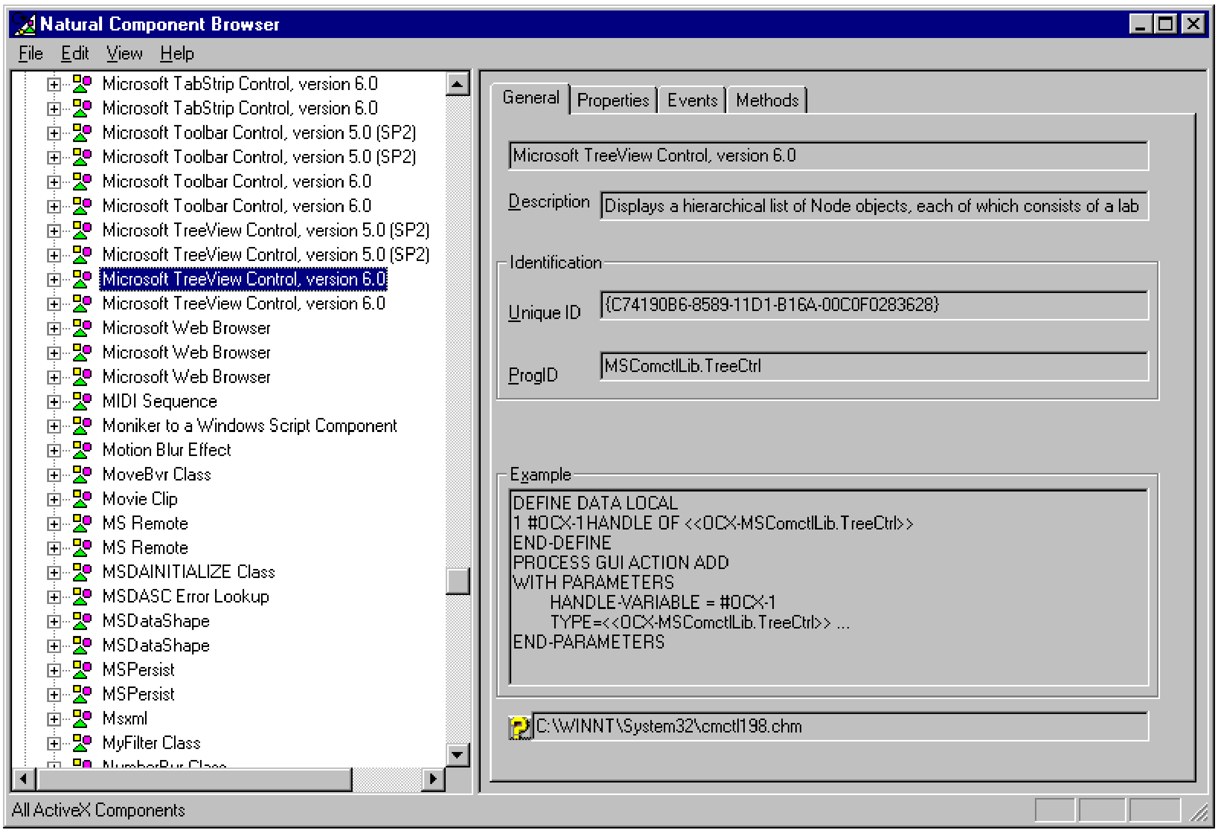
Group

If any of the group nodes ’All ActiveX Components’, ’ActiveX Controls’, ’Automation Objects’ or ’Interfaces’ is selected, the data view remains empty.

ActiveX Component

If a component node is selected, all four property pages are available. The page General provides the following information:

Name	**Component name**
Description	Short textual description.
Unique ID	Global Unique ID (GUID).
ProgID	ProgID.
Example	Natural statements that show how to use the component. (see Example Construction)
Help	Help file name. This file can be opened by pressing ?



Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
Msxml
MyFilter Class
NumberBvr Class

Microsoft TreeView Control, version 6.0

Description: Displays a hierarchical list of Node objects, each of which consists of a lab

Identification

Unique ID: {C74190B6-8589-11D1-B16A-00C0F0283628}

ProgID: MSComctlLib.TreeCtrl

Example

```
DEFINE DATA LOCAL
1 #OCX-1 HANDLE OF <<OCX-MSComctlLib.TreeCtrl>>
END-DEFINE
PROCESS GUI ACTION ADD
WITH PARAMETERS
     HANDLE-VARIABLE = #OCX-1
     TYPE=<<OCX-MSComctlLib.TreeCtrl>> ...
END-PARAMETERS
```

C:\WINNT\System32\cmctl198.chm

All ActiveX Components

If a component is selected and the page **Properties**, **Events** or **Methods** is activated, the information displayed on this page refers to the default interface.

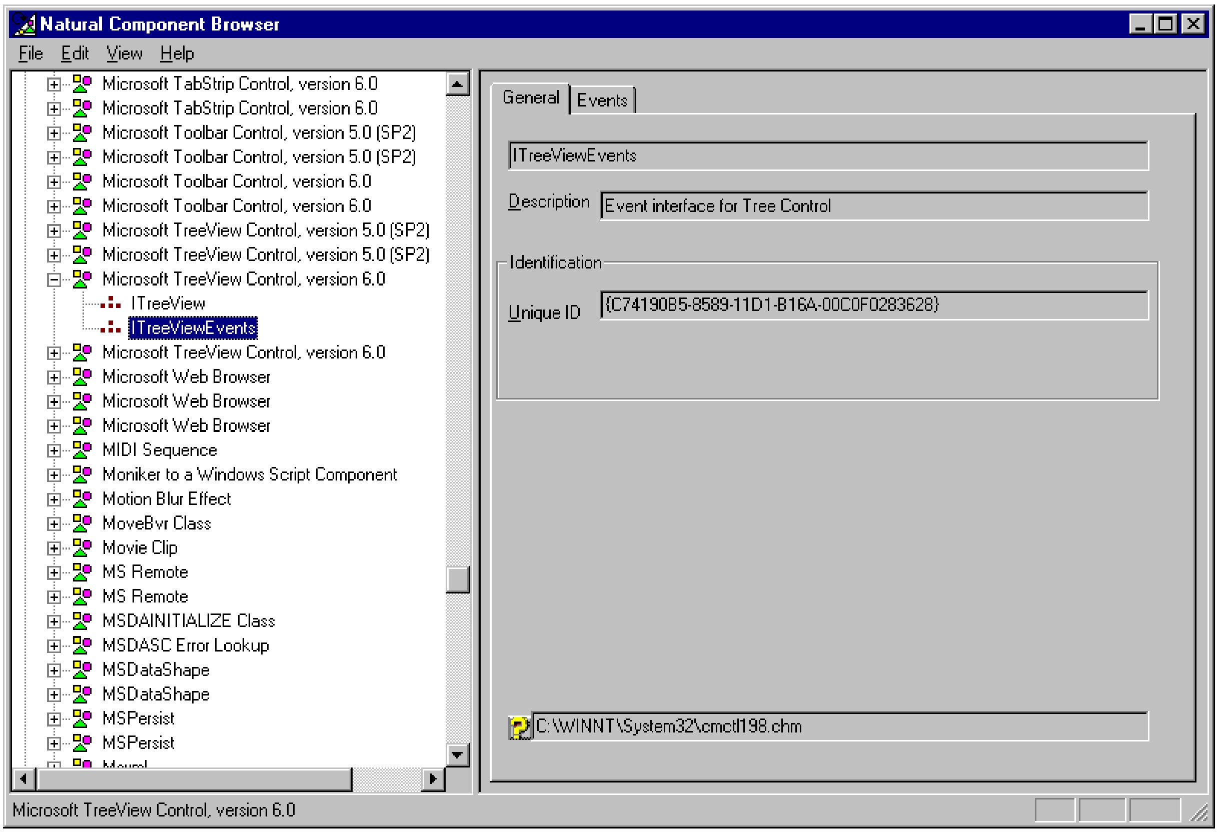
Interface

If an interface node is selected the number of available property pages depends on the type of the interface.

- If the Interface is browsed in the context of a component and if it is an Event Interface, then only the pages **General** and **Events** are set.
- Otherwise, the pages **General**, **Properties** and **Methods** are set.

In both cases, the page **General** provides the following information:

Name	**Interface name**
Description	Short textual description.
Unique ID	Global Unique ID (GUID).
Help	Help file name. This file can be opened by pressing


Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
ITreeView
ITreeViewEvents
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
General
Events
Identification
Microsoft TreeView Control, version 6.0

ITreeViewEvents

Description: Event interface for Tree Control

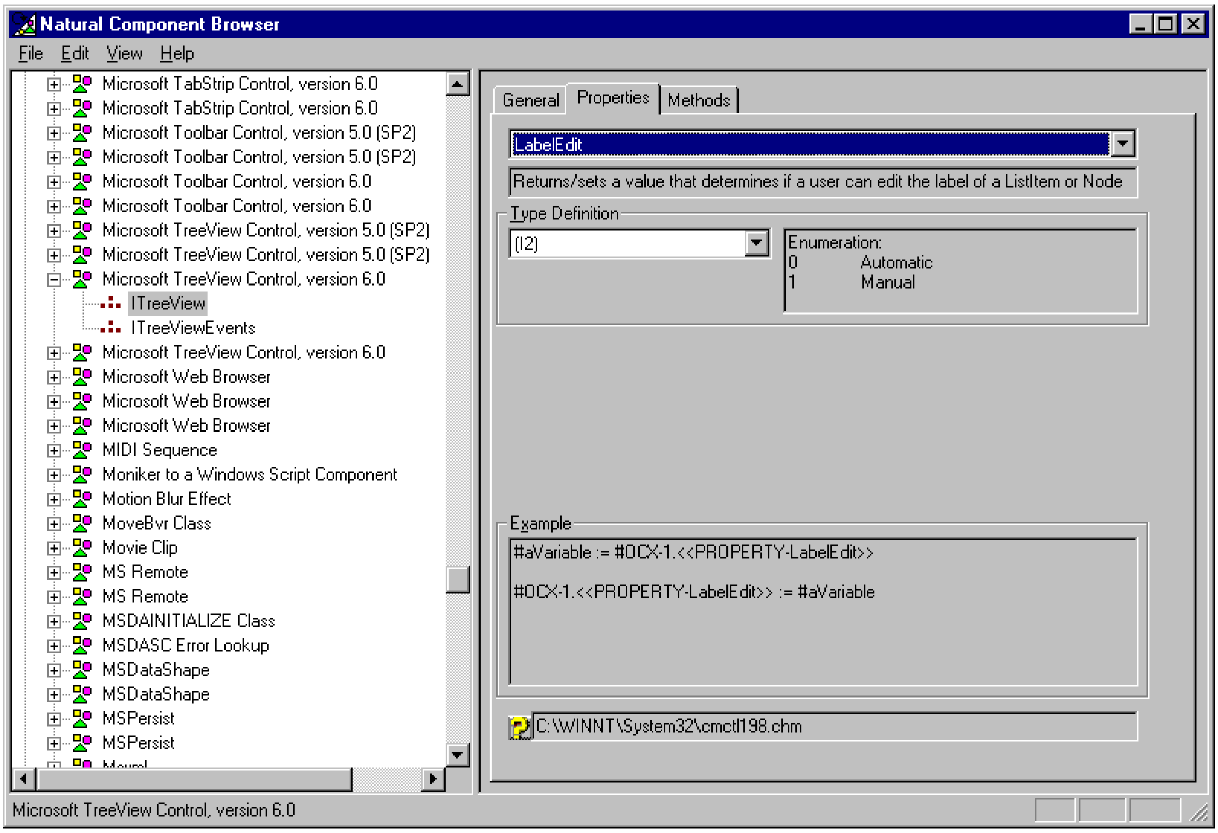
Unique ID: {C74190B5-8589-11D1-B16A-00C0F0283628}

C:\WINNT\System32\cmctl198.chm

Interfaces

The page **Properties** provides the list of properties that belong to the selected interface. For a specific property the following information is displayed:

Description	Short textual description for the selected property.
Type Definition	List of valid Natural formats for the property’s type. Additional type info on the selected Natural format, e.g. valid values for enumeration types.
Example	(see Example Construction).
Help	Help file name. This file can be opened by pressing


Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
ITreeView
ITreeViewEvents
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
General
Properties
Methods
Microsoft TreeView Control, version 6.0

LabelEdit

Returns/sets a value that determines if a user can edit the label of a ListItem or Node

Type Definition

(I2)

Enumeration:	
0	Automatic
1	Manual

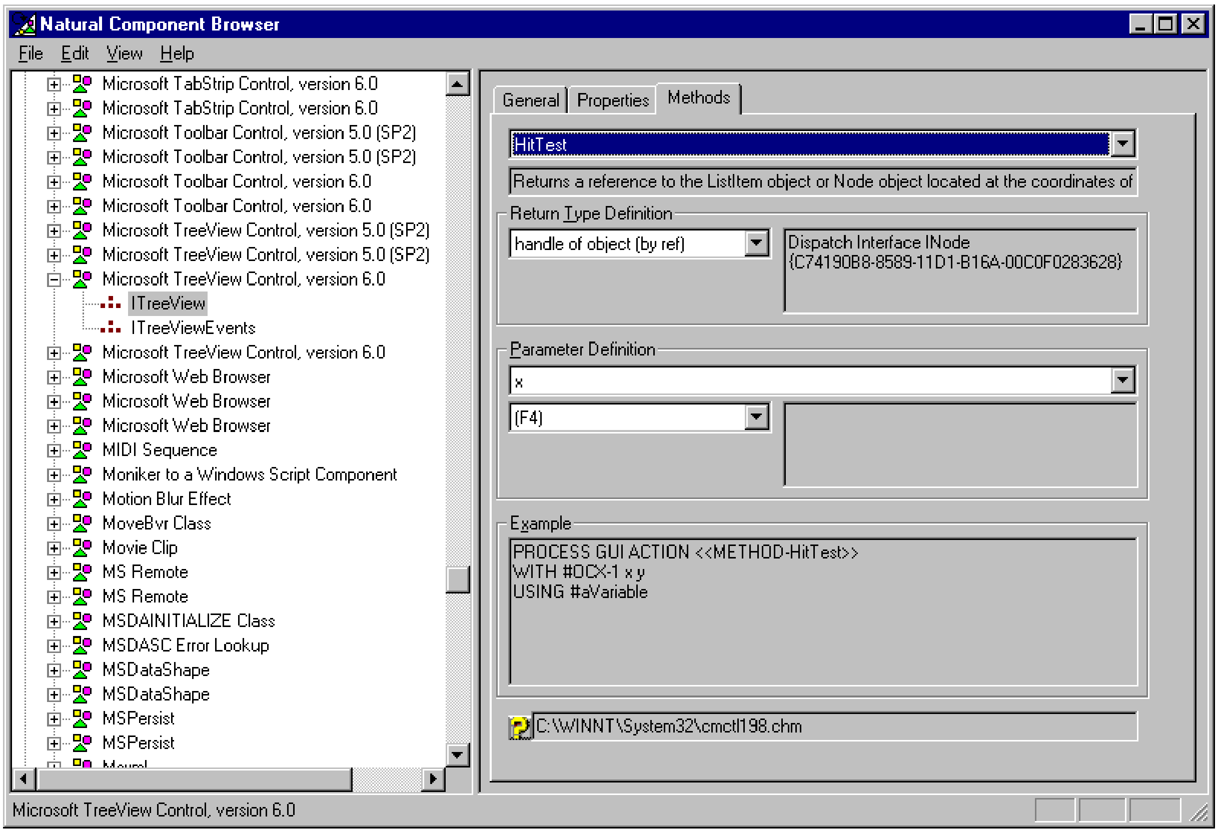
Example

```
#aVariable := #OCX-1.<<PROPERTY-LabelEdit>>

#OCX-1.<<PROPERTY-LabelEdit>> := #aVariable
```

C:\WINNT\System32\cmctl198.chm

The page **Methods** provides the list of methods that belong to the selected interface. This list includes GET-properties and PUT-properties. For a specific method or GET or PUT property the following information is displayed:

Description	Short textual description for the selected method.
Return Type Definition	List of valid Natural formats for the method’s type. Additional type info on a selected Natural format, e.g. Name and GUID of the interface if a handle of object corresponds to a dispatch interface. This interface can then be found in group ’Interfaces’.
Parameter Definition	List of parameters that are required by the method. List of Natural formats for each parameter together with additional info on the call mode (by ref). Additional type info on a selected Natural format, e.g. Name and GUID of the interface if a handle of object corresponds to a dispatch interface. This interface can then be found in group ’Interfaces’.
Example	(see Example Construction).
Help	Help file name. This file can be opened by pressing


Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
ITreeView
ITreeViewEvents
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
General
Properties
Methods

HitTest

Returns a reference to the ListItem object or Node object located at the coordinates of

Return Type Definition

handle of object (by ref)

Dispatch Interface INode
{C74190B8-8589-11D1-B16A-00C0F0283628}

Parameter Definition

x

(F4)

Example

```
PROCESS GUI ACTION <<METHOD-HitTest>>
WITH #OCX-1 x y
USING #aVariable
```

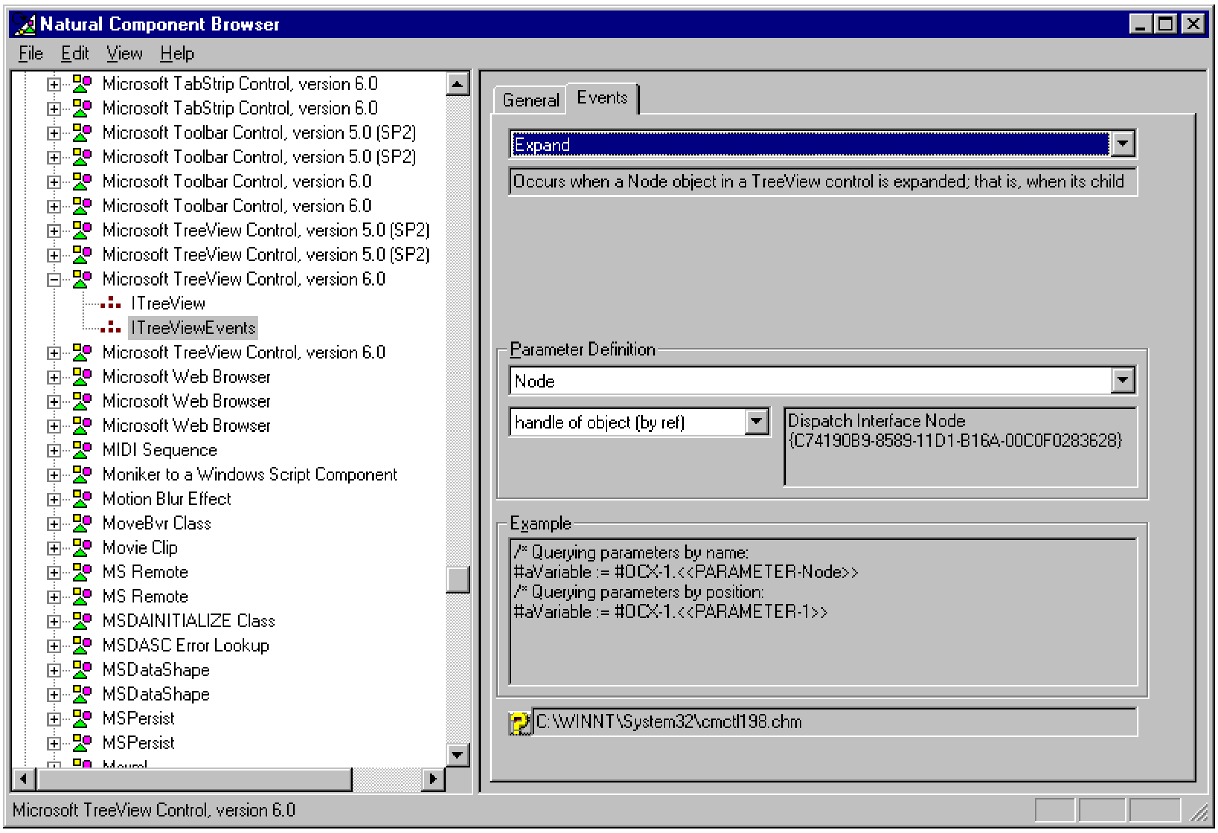
C:\WINNT\System32\cmctl198.chm

Microsoft TreeView Control, version 6.0

Event Interfaces

The page **Events** provides the list of events that belong to the selected interface. For a specific event the following information is displayed:

Description	Short textual description for the selected event.
Parameter Definition	List of parameters that are required by the event. List of valid Natural formats for each parameter together with additional info on the call mode (by ref). Additional type info on selected Natural format, e.g. Name and GUID of the interface if a handle of object corresponds to a dispatch interface. This interface can then be found in group ’Interfaces’.
Example	(see Example Construction).
Help	Help file name. This file can be opened by pressing


Natural Component Browser
File
Edit
View
Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
ITreeView
ITreeViewEvents
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
General
Events
Parameter Definition
Example

Expand

Occurs when a Node object in a TreeView control is expanded; that is, when its child

Node

handle of object (by ref)

Dispatch Interface Node
{C74190B9-8589-11D1-B16A-00C0F0283628}

```
/* Querying parameters by name:
#aVariable := #OCX-1.<<PARAMETER-Node>>
/* Querying parameters by position:
#aVariable := #OCX-1.<<PARAMETER-1>>
```

C:\WINNT\System32\cmctl198.chm

Microsoft TreeView Control, version 6.0

Menu

File

Item	Description
EXIT	Leaves the Component Browser.

Edit

Item	Description
Copy	This item is enabled if the data view has the focus. The selected text is copied to the clipboard.
Copy CLSID to Clipboard	This item is enabled if either the tree view or the data view has the focus. A component's CLSID is copied to the clipboard.
Copy ProgID to Clipboard	This item is enabled if either the tree view or the data view has the focus. A component's ProgID is copied to the clipboard.

View

Item	Description
Show by ProgID	This item is enabled if the tree view has the focus. By default, the tree view is sorted according to the component's external names. If this option is checked, it is sorted according to their ProgIDs. The currently displayed information is updated.
Show Current Version	This item is enabled if the tree view has the focus. By default, the tree view shows all versions of a component. If this option is checked, only the current version of a component is shown. The currently displayed information is updated.
Status Bar	Shows or hides the status bar.
Refresh	This item is enabled if the tree view has the focus. The tree view is refreshed, i.e. the information currently displayed is updated.

Help

Item	Description
About Component Browser	This item is enabled if either the tree view or the data view has the focus. Information on copyrights, version etc. is displayed.

Application Development Support

Example Construction

The data view provides detailed examples of Natural code that show how to use a selected object in a Natural application. Which statements are generated depends on the object’s type.

The example code or just parts of it can be selected, copied to the clipboard and used directly in an application. Only variable names might have to be adapted to meet the application’s requirements.

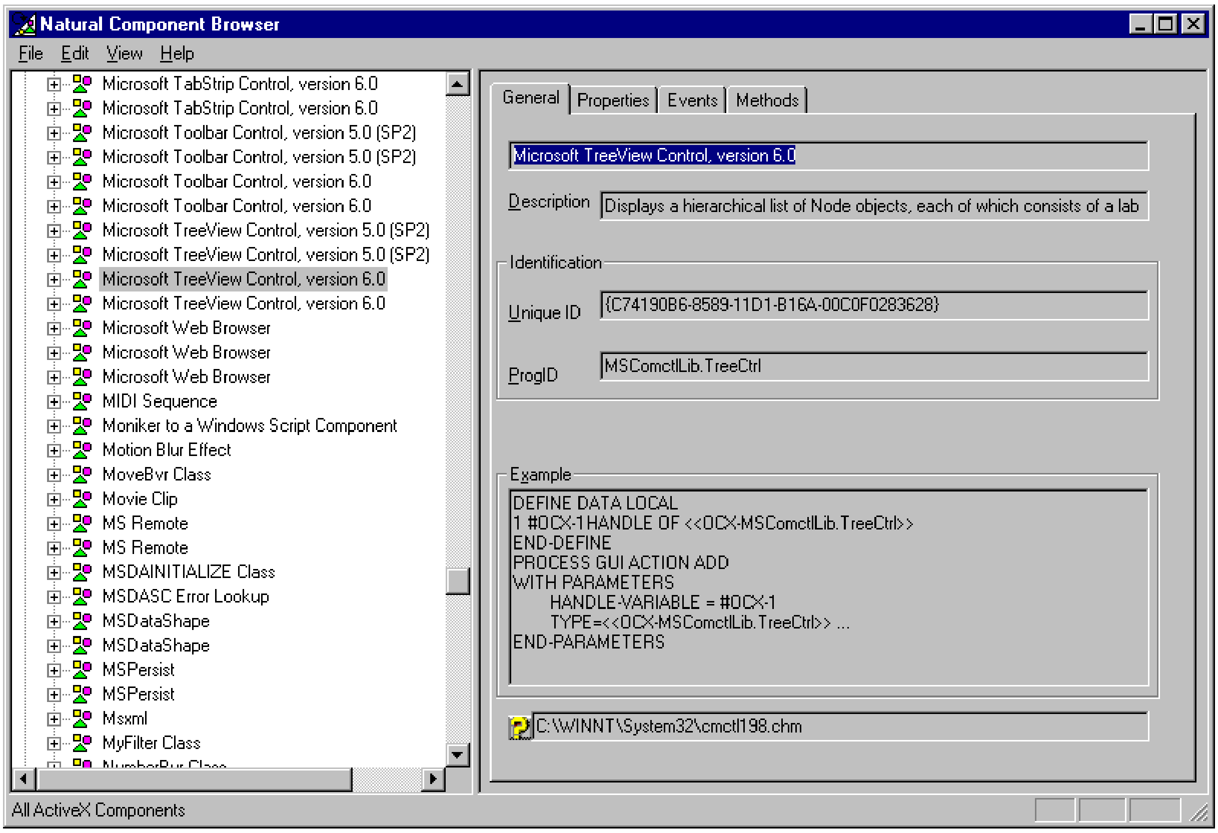
Frequently used identifiers such as CLSID and ProgID can be copied directly to the clipboard using menu item ’Edit > Copy CLSID to Clipboard’ or ’Edit > Copy ProgID to Clipboard’.

ActiveX Controls

For ActiveX controls, the appropriate PROCESS GUI statement is generated that shows how to use these components in Natural applications.

General

The example on the page **General** shows how an object of this type is instantiated. Here #OCX-1 denotes a variable that can be adapted to the current application.


Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
Msxml
MyFilter Class
General
Properties
Events
Methods
All ActiveX Components

Microsoft TreeView Control, version 6.0

Description: Displays a hierarchical list of Node objects, each of which consists of a lab

Identification

Unique ID: {C74190B6-8589-11D1-B16A-00C0F0283628}

ProgID: MSComctlLib.TreeCtrl

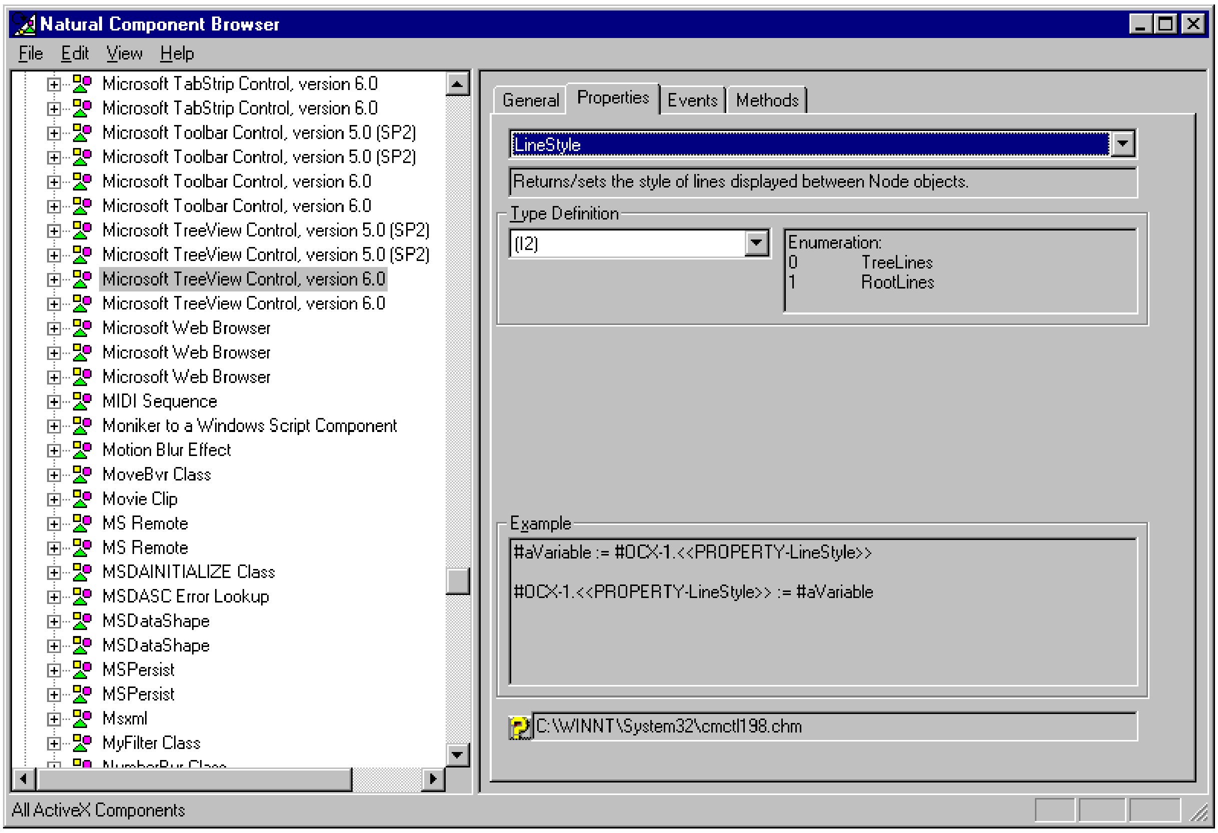
Example

```
DEFINE DATA LOCAL
1 #OCX-1 HANDLE OF <<OCX-MSComctlLib.TreeCtrl>>
END-DEFINE
PROCESS GUI ACTION ADD
WITH PARAMETERS
      HANDLE-VARIABLE = #OCX-1
      TYPE=<<OCX-MSComctlLib.TreeCtrl>> ...
END-PARAMETERS
```

C:\WINNT\System32\cmctl198.chm

Properties

The example on the page **Properties** shows how to assign a property to a variable and how to assign a variable to a property. Here #OCX-1 is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.



Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
Msxml
MyFilter Class
General
Properties
Events
Methods
Type Definition
Example
All ActiveX Components

LineStyle

Returns/sets the style of lines displayed between Node objects.

Type Definition: (I2)

Enumeration:
0 TreeLines
1 RootLines

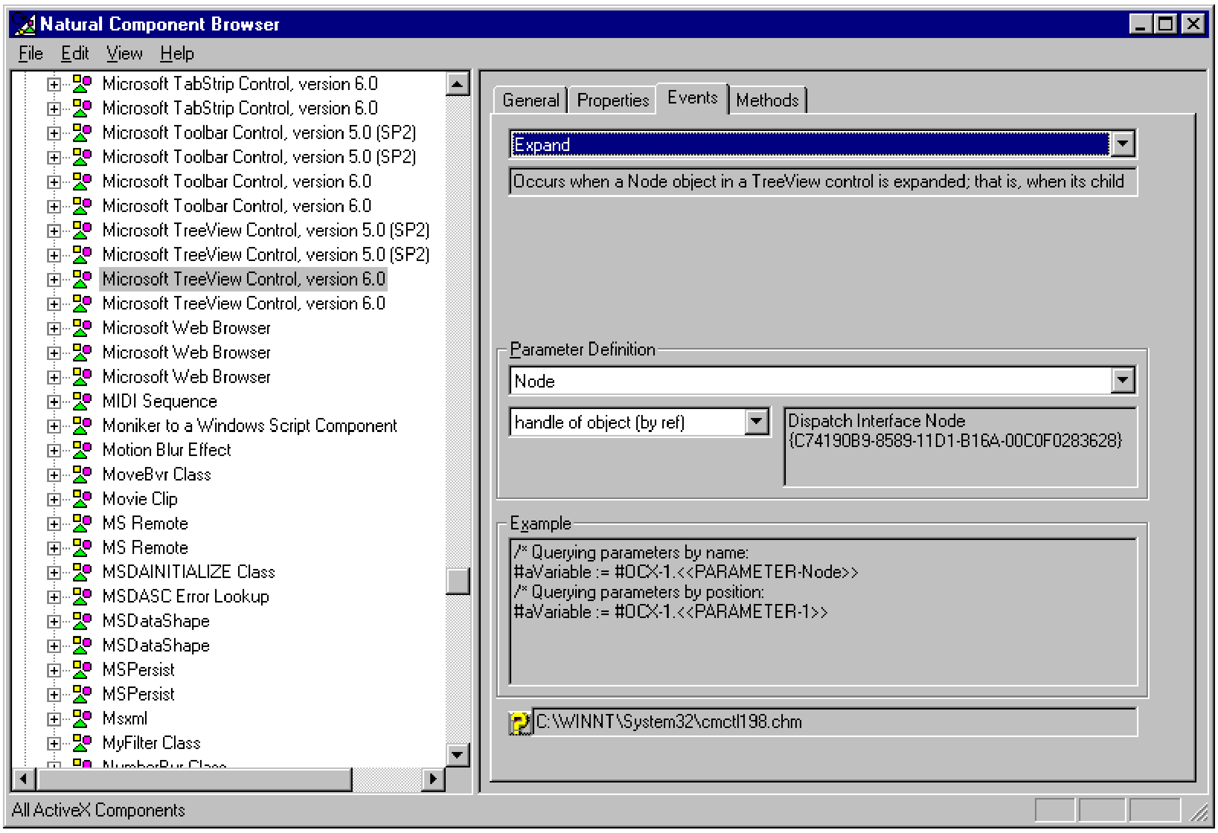
Example:

```
#aVariable := #OCX-1.<<PROPERTY-LineStyle>>

#OCX-1.<<PROPERTY-LineStyle>> := #aVariable
```

C:\WINNT\System32\cmctl198.chm

Events

The example on the page **Events** shows how to query event parameters by name or by position. Here #OCX-1 is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.


Natural Component Browser
File
Edit
View
Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
Msxml
MyFilter Class
NumberBvr Class
General
Properties
Events
Methods
Parameter Definition
Example
All ActiveX Components

Expand

Occurs when a Node object in a TreeView control is expanded; that is, when its child

Node

handle of object (by ref)

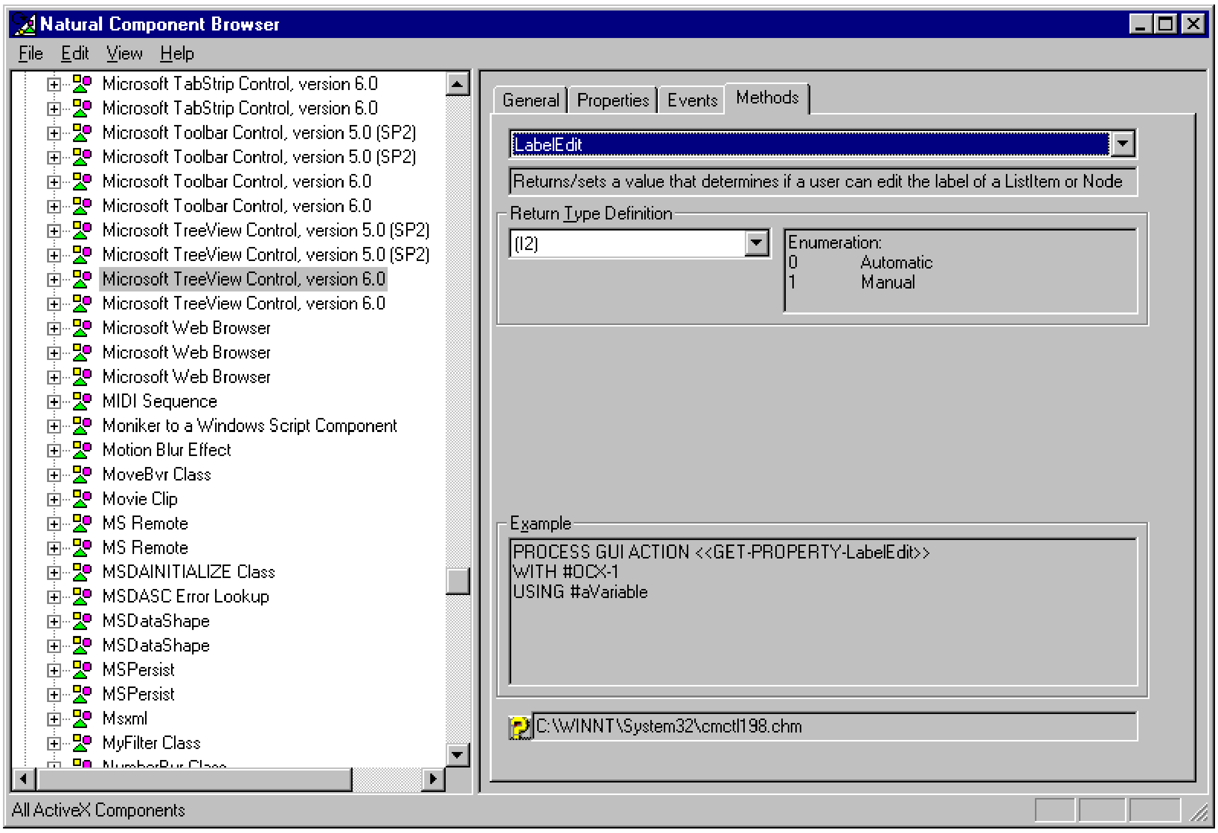
Dispatch Interface Node
{C74190B9-8589-11D1-B16A-00C0F0283628}

```
/* Querying parameters by name:
#aVariable := #OCX-1.<<PARAMETER-Node>>
/* Querying parameters by position:
#aVariable := #OCX-1.<<PARAMETER-1>>
```

C:\WINNT\System32\cmctl198.chm

Methods

The example on the page **Methods** shows how to use methods, GET-properties and PUT-properties. Here #OCX-1 is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required. The actual parameter names are already inserted into the statement if they are available. Otherwise default parameter names P0, P1, ... are used as placeholders. These names can be replaced by application-specific variables.


Natural Component Browser
File Edit View Help
Microsoft TabStrip Control, version 6.0
Microsoft TabStrip Control, version 6.0
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 5.0 (SP2)
Microsoft Toolbar Control, version 6.0
Microsoft Toolbar Control, version 6.0
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 5.0 (SP2)
Microsoft TreeView Control, version 6.0
Microsoft TreeView Control, version 6.0
Microsoft Web Browser
Microsoft Web Browser
Microsoft Web Browser
MIDI Sequence
Moniker to a Windows Script Component
Motion Blur Effect
MoveBvr Class
Movie Clip
MS Remote
MS Remote
MSDAINITIALIZE Class
MSDASC Error Lookup
MSDataShape
MSDataShape
MSPersist
MSPersist
Msxml
MyFilter Class
General
Properties
Events
Methods
Return Type Definition
Example
All ActiveX Components

LabelEdit

Returns/sets a value that determines if a user can edit the label of a ListItem or Node

(I2)

```
Enumeration:
0          Automatic
1          Manual
```

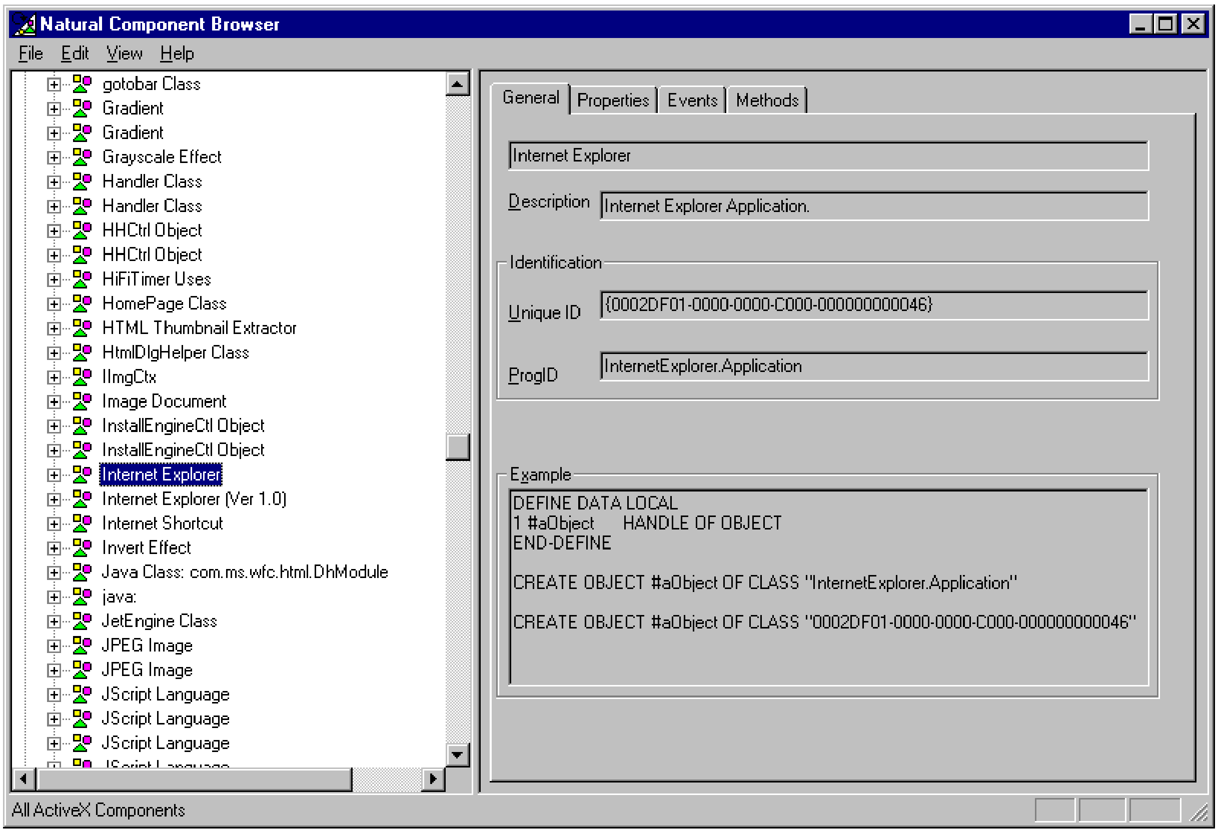
```
PROCESS GUI ACTION <<GET-PROPERTY-LabelEdit>>
WITH #OCX-1
USING #aVariable
```

C:\WINNT\System32\cmctl198.chm

Automation Objects

General

The example on the page **General** shows how an object of this type is instantiated. Here #aObject denotes a variable that can be adapted to the current application.



Internet Explorer

Description: Internet Explorer Application.

Identification

Unique ID: {0002DF01-0000-0000-C000-000000000046}

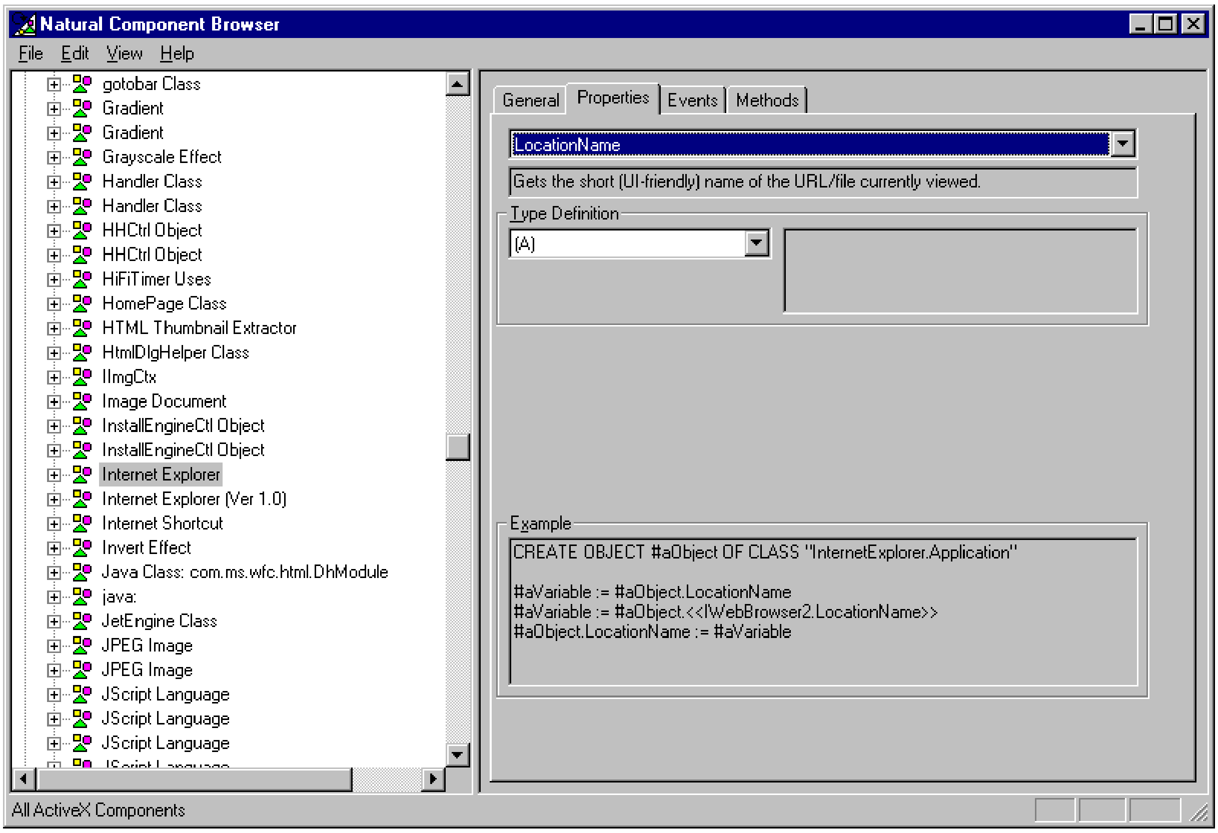
ProgID: InternetExplorer.Application

Example

```
DEFINE DATA LOCAL
1 #aObject      HANDLE OF OBJECT
END-DEFINE

CREATE OBJECT #aObject OF CLASS "InternetExplorer.Application"

CREATE OBJECT #aObject OF CLASS "0002DF01-0000-0000-C000-000000000046"
```

Properties

The example on the page **Properties** shows how to assign a property to a variable and how to assign a variable to a property. Here #aObject is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.



LocationName

Gets the short (UI-friendly) name of the URL/file currently viewed.

Type Definition: (A)

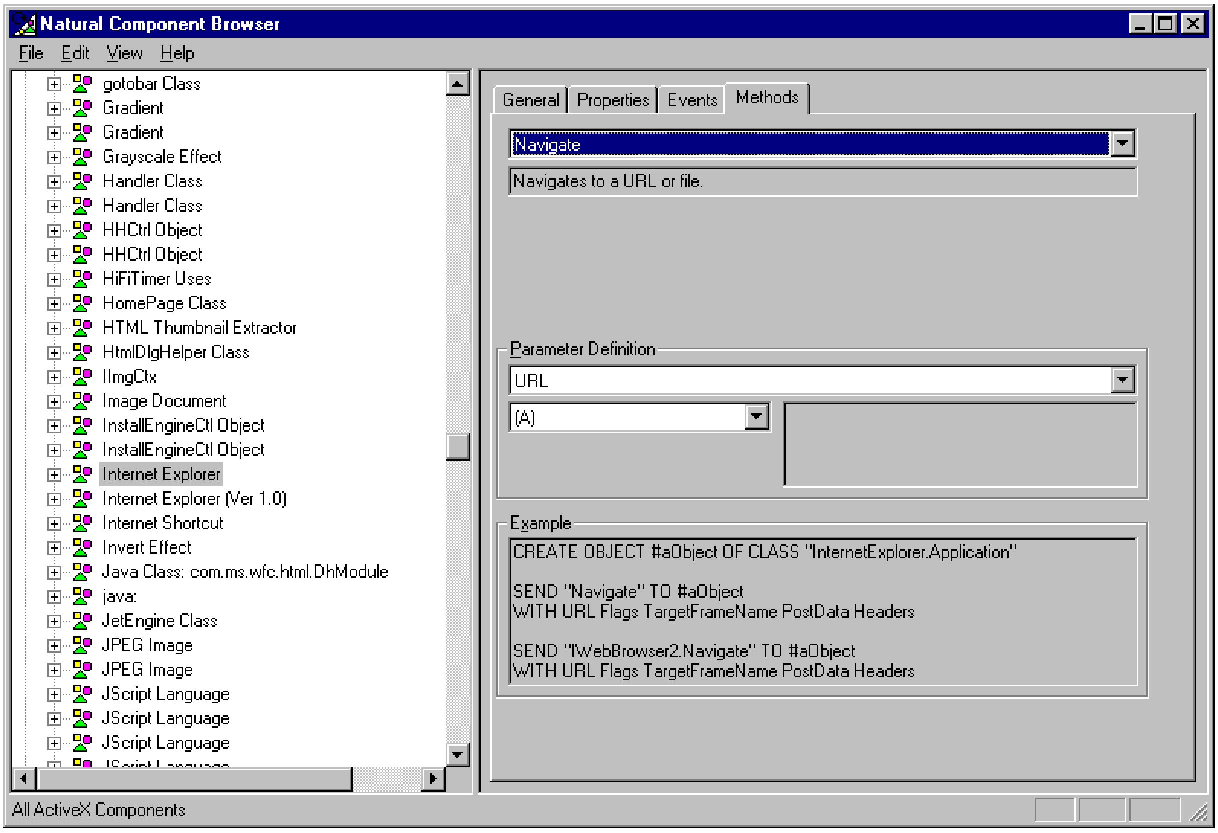
Example:

```
CREATE OBJECT #aObject OF CLASS "InternetExplorer.Application"

#aVariable := #aObject.LocationName
#aVariable := #aObject.<<IWebBrowser2.LocationName>>
#aObject.LocationName := #aVariable
```

Methods

The example on the page **Methods** shows how to use methods. Here #aObject is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.

The actual parameter names are already inserted into the statement if they are available. Otherwise default parameter names P0, P1, ... are used as placeholders. These names can be replaced by application-specific variables.



Navigate

Navigates to a URL or file.

URL

(A)

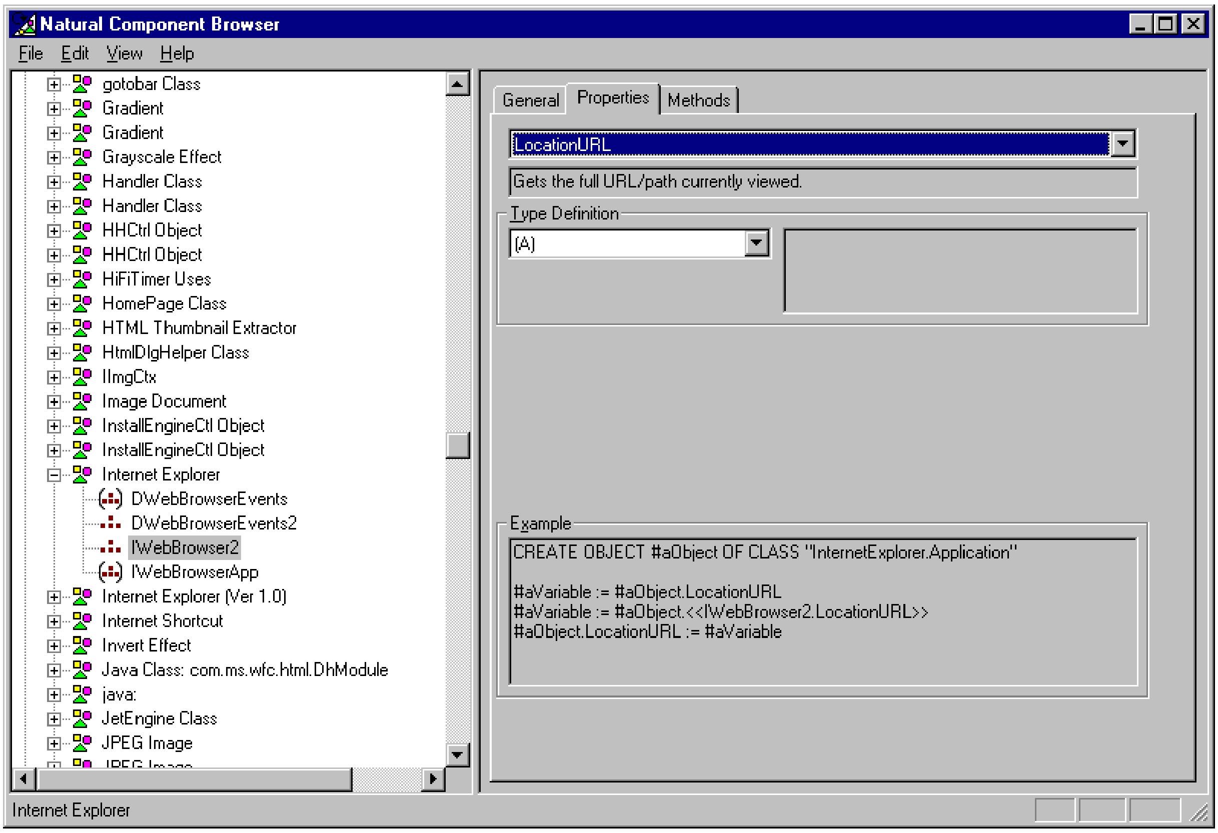
```
CREATE OBJECT #aObject OF CLASS "InternetExplorer.Application"

SEND "Navigate" TO #aObject
WITH URL Flags TargetFrameName PostData Headers

SEND "IWebBrowser2.Navigate" TO #aObject
WITH URL Flags TargetFrameName PostData Headers
```

Interfaces

For interfaces that belong to group ’Interfaces’ and that are not considered in the context of a class, the examples are generated as for Automation Objects. Only the CREATE OBJECT statement is left aside.

Properties

The example on the page **Properties** shows how to assign a property to a variable and how to assign a variable to a property. Here #aObject is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.



LocationURL

Gets the full URL/path currently viewed.

Type Definition: (A)

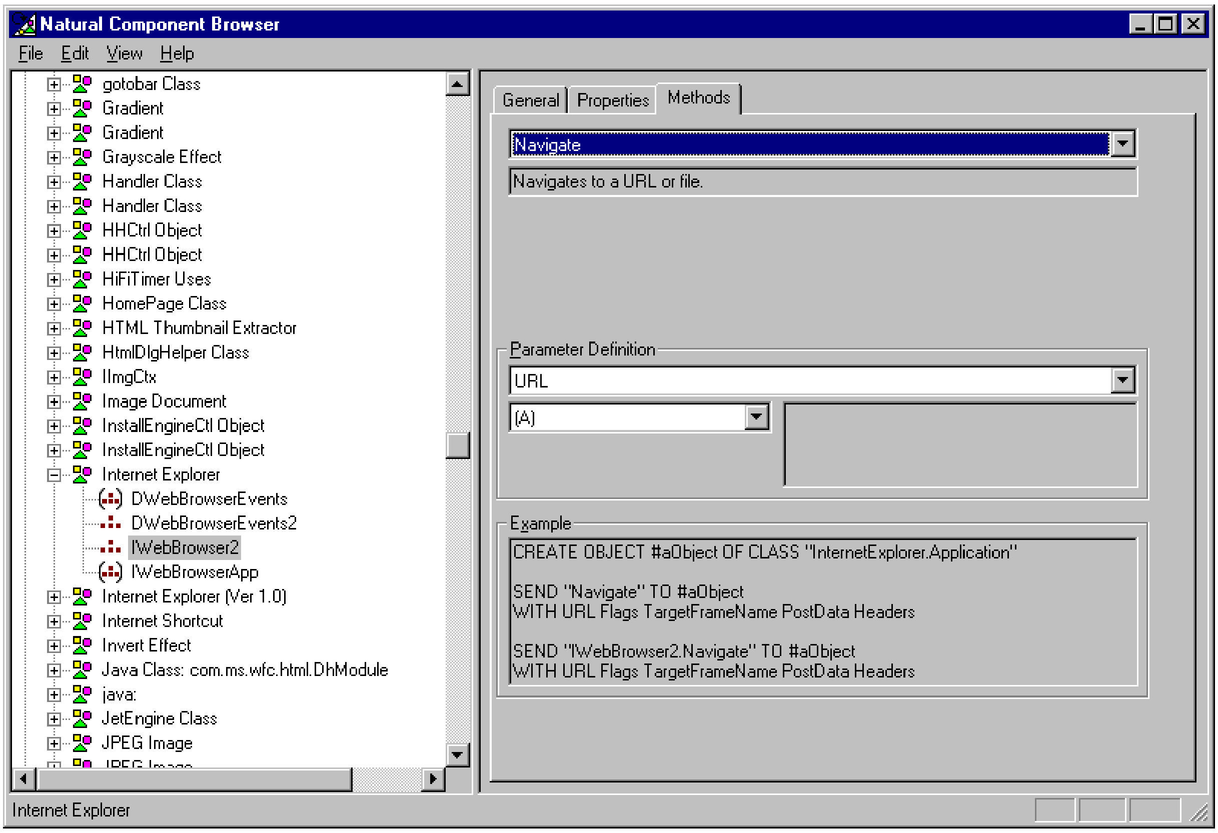
Example:

```
CREATE OBJECT #aObject OF CLASS "InternetExplorer.Application"

#aVariable := #aObject.LocationURL
#aVariable := #aObject.<<IWebBrowser2.LocationURL>>
#aObject.LocationURL := #aVariable
```

Methods

The example on the page **Methods** shows how to use methods. Here #aObject is the defined object handle and #aVariable refers to an application-specific variable. Both names can be adapted if required.

The actual parameter names are already inserted into the statement if they are available. Otherwise default parameter names P0, P1, ... are used as placeholders. These names can be replaced by application-specific variables.



Navigates to a URL or file.

```
CREATE OBJECT #aObject OF CLASS "InternetExplorer.Application"

SEND "Navigate" TO #aObject
WITH URL Flags TargetFrameName PostData Headers

SEND "IWebBrowser2.Navigate" TO #aObject
WITH URL Flags TargetFrameName PostData Headers
```

Plug-In Manager

The Natural Studio user interface is extensible by plug-ins. Part of the Natural Studio functionality itself is delivered in the form of plug-ins.

Once a plug-in is installed in Natural Studio, it needs not to be active and available in every Natural session and for every user. Which plug-ins are actually active and visible is configurable on a per user basis. The information which plug-ins are active for a user is stored in the user's profile in the registry. The user selects plug-ins for activation and deactivation with the Plug-in Manager.

The activation of plug-ins can also be entirely disabled and enabled on user basis. Initially, plug-in activation is disabled. In order to work with plug-ins, you must first enable plug-in activation.

Note:
A sample plug-in is delivered in source code in the library SYSEXPLG. With the current version of Natural Studio the plug-in interface is not released for external use. The interface will be further extended and possibly modified in upcoming versions. With the current version, you are therefore not recommended to implement your own plug-ins.

The following topics are covered:

- Purpose of the Plug-In Manager
- Enabling Plug-In Activation
- Activating the Plug-In Manager
- Deactivating the Plug-In Manager
- Using the Plug-In Manager
- Natural Studio Sample Plug-In

Purpose of the Plug-In Manager

The Plug-in Manager lists all installed plug-ins and shows their name, type, activation status and activation mode. In addition, it enables you to activate and deactivate installed plug-ins in order to configure your personal development environment.

Enabling Plug-In Activation

The activation of plug-ins can be entirely enabled and disabled on a per user basis. As default, plug-in activation is disabled. In order to work with plug-ins, you must first enable plug-in activation.

To enable Plug-in activation

- Check the checkbox "Enable plug-ins" in **Tools** > **Options** > **Workspace**.

To disable Plug-In activation

- Uncheck the checkbox "Enable plug-ins" in **Tools** > **Options** > **Workspace**.

Activating the Plug-In Manager

The Plug-in Manager is itself implemented as a plug-in written in Natural. By definition, the Plug-in Manager itself cannot be dynamically activated. If plug-in activation is enabled, the Plug-in Manager is instead always available under the fixed item **Plug-in Manager** in the **Tools** menu bar.

Deactivating the Plug-In Manager

The Plug-in Manager is included in the list of plug-ins where it can be deactivated like any other plug-in (see To deactivate a plug-in).

But, of course, it can then not be reactivated manually in the same Natural Studio session, because in order to do so, you would need an active Plug-in Manager. However, as its activation mode is Automatic, it will be activated again at the start of the next Natural Studio session if plug-in activation is enabled.

Using the Plug-In Manager

If you do not see the **Tools** menu bar, first make it visible by selecting **Tools** > **Customize** > **Toolbars**.

To start the Plug-in Manager

- Click its icon in the **Tools** menu bar.

The Plug-in Manager displays the list of plug-ins available in an installation and shows them in a list. The plug-ins that are already active for you are marked. You may activate or deactivate plug-ins.

Using context menu commands, you can modify the activation status and mode of individual Plug-ins.

To activate a plug-in

1. Select the plug-in in the list.
2. Click the right mouse button.
3. From the resulting context menu, choose **Activate**.

To deactivate a plug-in

1. Select the plug-in in the list.
2. Click the right mouse button.
3. From the resulting context menu, choose **Deactivate**.

To cause the plug-in to be activated each time you start Natural Studio

1. Select the plug-in in the list.
2. Click the right mouse button.
3. From the resulting context menu, choose **Activation mode** > **Automatic**.

To cause the plug-in to stay inactive when you start Natural Studio

1. Select the plug-in in the list.
2. Click the right mouse button.
3. From the resulting context menu, choose **Activation mode** > **Manual**.
   You can later activate the plug-in manually using the Plug-in Manager.

Natural Studio Sample Plug-In

- Purpose of the Natural Studio Sample Plug-In
- Activating the Sample Plug-In
- Using the Sample Plug-In
- Deactivating the Sample Plug-In

Purpose of the Natural Studio Sample Plug-In

The Natural Studio Sample Plug-in demonstrates how the Natural Studio metastructure can be extended with plug-ins that define your own object types, assign commands to them and display objects as nodes in tree views and list views.

The sample plug-in shows information about the Natural User Exit subprograms contained in the library SYSEXT. It allows users to list their documentation and to execute a test program for each of them.

The source code of the sample plug-in (contained in the library SYSEXPLG) is intended to give an impression of how plug-ins can be implemented with Natural Studio. With the current version of Natural Studio the plug-in interface itself is not released for external use. With this version, it is therefore not recommended to use the interface to implement your own plug-ins.

Activating the Sample Plug-In

The sample plug-in is installed automatically during Natural Studio installation. By default the activation of plug-ins is disabled. Therefore in order to use the sample plug-in, you must first enable plug-in activation and then activate the sample plug-in.

To enable Plug-in activation

- Check the checkbox "Enable plug-ins" in **Tools** > **Options** > **Workspace**.

To activate the sample plug-in:

1. Start the Plug-in Manager using **Tools** > **Configuration Tools** > **Plug-in Manager**.
2. Select "Natural Studio Sample Plug-in" in the "Plug-in Manager" window.
3. Select the **Activate** command in the context menu
4. Close the Plug-in Manager

Using the Sample Plug-In

During activation, the plug-in creates and displays a toolbar that contains two commands: "Open Tree View" and "Open List View". These two commands also appear in the **Sample Plug-in** sub-menu in the **Tools** menu and in the context menus of the Natural object types Program, Subprogram and Text.

Command "Open Tree View":

- If a user exit subprogram (subprogram USRnnnnN in library SYSEXT), its description (text member USRnnnnT in library SYSEXT) or its test program (program USRnnnnP in library SYSEXT) is selected, the command displays information about this user exit in a tree view.
- If none of the above is selected, the command displays information about all user exit in a tree view.

Command "Open List View":

- Displays the same information in a list view

Deactivating the Sample Plug-In

To deactivate the sample plug-in:

1. Start the Plug-in Manager using **Tools** > **Configuration Tools** > **Plug-in Manager**.
2. Select "Natural Studio Sample Plug-in" in the "Plug-in Manager" window
3. Select **Deactivate** in the context menu.
4. Close the Plug-in Manager

Large and Dynamic Variables/Fields

The following topics are covered below:

- Introduction
- Definition of Dynamic Variables
- System Variable *LENGTH - field
- Size Limitation Checks
- Statements EXPAND and REDUCE
- Usage of Dynamic Variables

Introduction

Since version 4.1 Natural for Windows provides enhanced capabilities for the usage of large variables by removing the existing size limitations and by providing for dynamic allocation of these variables at execution time.

Large variables for alpha and binary data are based on the well known Natural formats A and B. The limitations of 253 for format A and 126 for format B are no longer in effect. The new size limit is 1 GB. These large static variables and fields are handled in the same manner as traditional alpha and binary variables and fields with regard to definition, redefinition, value space allocation, conversions, referencing in statements, etc. All rules concerning alpha and binary formats apply to these large formats.

In that the maximum size of large data structures (for example, pictures, sounds, videos) may not exactly be known at application development time, Natural additionally provides for the definition of alpha and binary variables with the attribute DYNAMIC. The value space of variables which are defined with this attribute will be extended dynamically at execution time when it becomes necessary (for example, during an assignment operation: #picture1 := #picture2). This means that large binary and alpha data structures may be processed in Natural without the need to define a limit at development time. The execution-time allocation of dynamic variable is of course subject to available memory restrictions. If the allocation of dynamic variables results in an insufficent memory condition being returned by the underlying operating system, the ON ERROR statement can be used to intercept this error condition; otherwise, an error message will be returned by Natural.

The new Natural system variable *LENGTH can be used to obtain the number of bytes of the value space which are currently used for a given dynamic variable. Natural automatically sets *LENGTH to the length of the source operand during assignments in which the dynamic variable is involved. *LENGTH(field) therefore returns the size currently used for a dynamic Natural field or variable in bytes.

If the dynamic variable space is no longer needed, the REDUCE DYNAMIC VARIABLE statement can be used to reduce the space used for the dynamic variable to zero (or any other desired size). If the upper limit of memory usage is known for a specific dynamic variable, the EXPAND statement can be used to set the space used for the dynamic variable to this specific size.

If a dynamic variable is to be initilialized, the MOVE ALL UNTIL statement should be used for this purpose.

Definition of Dynamic Variables

Because the actual size of large alpha and binary data structures may not be exactly known at application development time, the definition of ***dynamic*** variables of formats A or B can be used to manage these structures. The dynamic allocation and extension (reallocation) of large variables is transparent to the application programming logic. Dynamic variables are defined without any length. Memory allocation will be done at execution time implicitly, when the dynamic variable is used as a target operand or explicitly with an EXPAND statement.

Dynamic variables can only be defined in a DEFINE DATA statement using the following syntax:

```
level variable-name ( A ) DYNAMIC
level variable-name ( B ) DYNAMIC
```

A dynamic variable can only be defined as **scalar** (no dynamic array definition is possible).
A dynamic variable may not be contained in a **REDEFINE clause** and a **redefinition** of a dynamic variable or of a group that contains a dynamic variable is not possible. The **CONST** and **INIT** clauses are invalid for dynamic variables.

System Variable *LENGTH - field

The size of the currently used value space of a dynamic variable can be obtained from the system variable *LENGTH. *LENGTH is set to the (used) length of the source operand during assignments automatically.

Note: Due to performance considerations, the allocated size may be larger than the used size. It is not possible for the Natural programmer to obtain information about the currently allocated size. This is an internal value.

*LENGTH(field) returns the used size of a dynamic Natural field or variable in bytes. *LENGTH may be used only for dynamic variables to get the currently used size.

Size Limitation Checks

Profile Parameter DSLM

For compatibility reasons, a size limitation check at compile time for fixed length variables can be done using the DSLM parameter. The DSLM parameter limits the size to 32 KB per variable.

Profile Parameter USIZE

For dynamic variables, a size limitation check at compile time is not possible because no length is defined for dynamic variables. The Size of User Buffer Area (USIZE) indicates the size of the user buffer in virtual memory. The user buffer contains all data dynamically allocated by Natural. If a dynamic variable is allocated or extended at execution time and the USIZE limitation is exceeded, an error message will be returned.

Statements EXPAND and REDUCE

The statements EXPAND and REDUCE are used to explicitly allocate and free memory space for a dynamic variable.

Syntax:

```
EXPAND [ SIZE OF ] DYNAMIC [ VARIABLE ] operand1 TO operand2
REDUCE [ SIZE OF ] DYNAMIC [ VARIABLE ] operand1 TO operand2
```

where operand1 is a dynamic variable and operand2 is a non-negative numeric size value.

The EXPAND statement is used to extend the allocated size of the dynamic variable to a given size. The size currently used (*LENGTH) for the dynamic variable is not modified.
If the given size is less than the currently allocated size of the dynamic variable, the statement will be ignored.

The REDUCE statement is used to reduce the allocated memory size. The allocated memory of the dynamic variable beyond the given size is released immediately (when the statement is executed).
If the size currently used (*LENGTH) for the dynamic variable is greater than the given size, *LENGTH of this

dynamic variable is set to this size. The content of the variable is truncated, but not modified. If the given size is larger as the currently allocated size of the dynamic variable, the statement will be ignored.

Usage of Dynamic Variables

- Assignments with Dynamic Variables
- Initialization of Dynamic Variables
- String Manipulation with Dynamic Alpha Variables
- Logical Condition Criterion - LCC - with Dynamic Variables
- Parameter Transfer with Dynamic Variables
- Work File Access with Large and Dynamic Variables
- DDM Generation and Editing for Varying Length Columns
- Accessing Large Database Objects
- Performance Aspects with Dynamic Variables

Generally, a dynamic alpha variable may be used wherever an operand of format A or format B is allowed.

Exception:
Dynamic variables are not allowed within DISPLAY, WRITE, PRINT, STACK, INPUT, REINPUT, and SORT statements. (To DISPLAY, WRITE or PRINT dynamic alpha variables, the variable must be cut into smaller portions using the SUBSTRING statement.)

Exception 2:
Large and dynamic variables are not supported by the Natural remote procedure call.

The used length (*LENGTH) and the allocated size of dynamic variables are equal to zero until such time that the variable is first accessed as a target operand. Due to assignments or other manipulation operations, dynamic variables may be firstly allocated or extended (reallocated) to the exactly size value of the source operand.

The size of a dynamic variable may be extended if it is used as a modifiable operand (target operand) in the following statements:

- destination operand in an assignment (ASSIGN, MOVE)
- operand2 in COMPRESS
- operand1 in EXAMINE REPLACE
- operand4 in SEPARATE
- READ WORK FILE
- parameter or view field in the INTO clause of SELECT
- CALLNAT, PERFORM (except AD=O, or BY VALUE in PDA)
- SEND METHOD

Currently, there are the following limits concerning the usage of large variables:

- EntireX TCP/IP limit is 4MB
- Network limit is 32K
- CALL statement parameter size less than 65K per parameter (no limit for the CALL with INTERFACE4 option)

In the following sections the usage of dynamic variables are discussed in more detail with examples..

Assignments with Dynamic Variables

Generally, an assignment is done in the current used length (*LENGTH) of the source operand.
If the destination operand is a dynamic variable, its current allocated size is possibly extended in order to move the source operand without truncation.

Example:

```
#MyDynText1 := operand or
MOVE operand to #MyDynText1
#MyDynText1 is automatically extended until the source operand can be copied
```

MOVE ALL, MOVE ALL UNTIL with dynamic target operands are defined as follows:
MOVE ALL moves the source operand repeatedly to the target operand until the used length (*LENGTH) of the target operand is reached. *LENGTH is not modified. If *LENGTH is zero, the statement will be ignored.
MOVE ALL operand1 TO operand2 UNTIL operand3 moves operand1 repeatedly to operand2 until the length specified in operand3 is reached. If operand3 is greater than *LENGTH(operand2), operand2 is extended and *LENGTH(operand2) is set to operand3. If operand3 is less than *LENGTH(operand2), the used length is reduced to operand3. If operand 3 equals *LENGTH(operand2), the behavior is equivalent to MOVE ALL.

Example:

```
#MyDynText1 := 'ABCDEFGHIJKLMNO'     /* *LENGTH(#MyDynText1) is 15
MOVE ALL 'AB' TO #MyDynText1         /* content of #MyDynText1 is 'ABABABABABABABA';
                                     /* *LENGTH is still 15
MOVE ALL 'CD' TO #MyDynText1 UNTIL 6 /* content of #MyDynText1 is 'CDCDCD';
                                     /* *LENGTH is 6
MOVE ALL 'EF' TO #MyDynText1 UNTIL 10/* content of #MyDynText1 is 'EFEFEFEFEF';
                                     /* *LENGTH is 10
```

MOVE JUSTIFIED is rejected at compile time if the target operand is a dynamic variable.

MOVE SUBSTR and MOVE TO SUBSTR are allowed. MOVE SUBSTR will lead to runtime error if a sub-string behind the used length of a dynamic variable (*LENGTH) is referenced. MOVE TO SUBSTR will lead to runtime error if a sub-string position behind *LENGTH + 1 is referenced, because this would lead to an undefined gap in the content of the dynamic variable. If the target operand should be extended by MOVE TO SUBSTR (for example if the second operand is set to *LENGTH+1), the third operand is mandatory.

Example:

```
/* valid
#op2 := *LENGTH(#MyDynText1)
MOVE SUBSTR (#MyDynText1, #op2) TO operand                /* move last character to operand
#op2 := *LENGTH(#MyDynText1) + 1
MOVE operand TO SUBSTR(#MyDynText1, #op2, #len_operand)
                                                          /* concatenate operand to #MyDynText1
/* invalid
#op2 := *LENGTH(#MyDynText1) +1
MOVE SUBSTR (#MyDynText1, #op2, 10) TO operand            /* leads to runtime error; undefined sub-string
#op2 := *LENGTH(#MyDynText1 +10)
MOVE operand TO SUBSTR(#MyDynText1, #op2, #len_operand)
       /* leads to runtime error; undefined gap
#op2 := *LENGTH(#MyDynText1) +1
MOVE operand TO SUBSTR(#MyDynText1, #op2)                 /* leads to runtime error; undefined length
```

Assignment Compatibility

Example:

```
#MyDynText1   := #MyStaticVar1
#MyStaticVar1 := #MyDynText2
```

If the source operand is a static variable, the used length of the dynamic destination operand (*LENGTH(#MyDynText1)) is set to the format length of the static variable and the source operand is copied in this length including trailing blanks (format A) or zeros (format B).

If the destination operand is static and the source operand is dynamic, the dynamic variable is copied in its currently used size. If this size is less than the format length of the static variable, the rest is filled with blanks or zeros. Otherwise, the value will be truncated. If the currently used size of the dynamic variable is 0, the static target operand is filled with blanks or zeros.

Initialization of Dynamic Variables

Dynamic Variables can be initialized with blanks (alpha) or zeros (binary) up to the currently used length (= *LENGTH) using the RESET statement. *LENGTH is not modified.

Example:

```
DEFINE DATA LOCAL
:
#MyDynText1   (A)  DYNAMIC
:
END-DEFINE
:
#MyDynText1 := ’short text’
write *LENGTH(#MyDynText1)          /* used length is 10
:
RESET #MyDynText1                   /* used length is still 10, value is 10 blanks
:
```

To initialize a dynamic variable with a specified value in a specified size, the MOVE ALL UNTIL statement may be used.

Example:

```
:
MOVE ALL ’Y’ TO #MyDynText1 UNTIL 15          /* #MyDynText1 contains 15 ’Y’s, used length is 15
:
```

String Manipulation with Dynamic Alpha Variables

If a modifiable operand is a dynamic variable, its current allocated size is possibly extended in order to perform the operation without truncation or an error message. This is valid for the concatenation (COMPRESS) and separation of dynamic alpha variables (SEPARATE).

Example:

```
DEFINE DATA LOCAL
   1  #MyDynText1          (A)  DYNAMIC
   1  #MyDynText2          (A)  DYNAMIC
...
COMPRESS INTO
...
#MyDynText2

#MyDynText2 will be extended in order to compress the source operands.
Note: in case of non-dynamic variables the value may be truncated.

SEPARATE INTO #MyDynText1 #MyDynText2 WITH DELIMITER
#MyDynText1 and #MyDynText2 are possibly extended or reduced to separate the source operand.

EXAMINE #MyDynText1 FOR REPLACE
#MyDyntext1 will possibly be extended or reduced to perform the REPLACE operation successfully.
```

Note: in case of non-dynamic variables an error message may be returned.

Logical Condition Criterion - LCC - with Dynamic Variables

Generally, a read-only operation (such as LCC) with a dynamic variable is done with its current used size. Dynamic variables are processed like static variables if they are used in a read-only (non-modifiable) context.

Example:

```
IF #MyDynText1 = #MyDynText2 OR #MyDynText1 = "**"
IF #MyDynText1 < #MyDynText2 OR #MyDynText1 < "**"
IF #MyDynText1 > #MyDynText2 OR #MyDynText1 > "**"
```

Also in the case of trailing blanks or leading zeros, dynamic variables will show an equivalent behavior.
For dynamic variables the alpha value ’AA ’ will be equal to ’AA’ and the binary value ’00003031’ is equal to ’3031’. If a comparison result should only be TRUE in case of an exact copy, the used lengths of the dynamic variables have to be compared in addition. For example, a variable for a small music clip and a variable for the same clip with a following period of silence at the end could be compared. If one variable is an exactly copy of the other, their used lengths are also equal.

Example:

```
#MyDynText1  :=  ’hello’          /* used length is 5
#MyDynText2  :=  ’hello     ’     /* used length is 10
IF  #MyDynText1  =  #MyDynText2   /* TRUE
:
IF   #MyDynText1  =  #MyDynText2
  AND  *LENGTH(#MyDynText1) = *LENGTH(#MyDynText2)
                                 /* FALSE
:
```

Two dynamic variables are compared position by position from left to right up to the minimum of their used lengths. The first position where the variables are not equal determines if the first or the second variable is greater than, less than or equal to the other. The variables are equal if they are equal up to the minimum of their used lengths and the rest of the longer variable contains only blanks (format A) or zeros (format B).

Example:

```
#MyDynText1  :=  ’hello1’          /* used length is 6
#MyDynText2  :=  ’hello2     ’     /* used length is 10
IF  #MyDynText1   #MyDynText2      /* TRUE
: #MyDynText2
:="hallo" IF #MyDynText1>  #MyDynText2    /* TRUE
:
```

Comparison Compatibility

Comparisons between dynamic and static variables are equivalent to comparisons between dynamic variables. The format length of the static variable is interpreted as its used length.

Example:

```
#MyStatText1 :=  ’hello’            /* format length of MyStatText1 is 20
#MyDynText1  :=  ’hello’            /* used length is 5
IF  #MyStatText1  =  #MyDynText1    /* TRUE
:
IF  #MyStatText1  >  #MyDynText1    /* FALSE
```

Parameter Transfer with Dynamic Variables

Dynamic variables may be passed as parameters to a called program object (CALLNAT, PERFORM).
Call-by-reference is possible because the value space of a dynamic variable is contiguous. Call-by-value causes an assignment with the variable definition of the caller as the source operand and the parameter definition as the destination operand. Call-by-value result causes in addition the movement in the opposite direction
For call-by-reference, both definitions must be DYNAMIC. If only one of them is DYNAMIC, a runtime error is raised. In case of call-by-value (result) all combinations are possible. The following table illustrates the valid combinations:.

Call By Reference

	Parameter	
Caller	**Static**	**Dynamic**
Static	Yes	No
Dynamic	No	Yes

The formats of dynamic variables A or B must match.

Call by Value - Result

	Parameter	
Caller	**Static**	**Dynamic**
Static	Yes	Yes
Dynamic	Yes	Yes

Note: in case of static/dynamic or dynamic/static definitions a value truncation may occur according to the data transfer rules of the appropriate assignments.

Example 1:

```
DEFINE DATA LOCAL
    1 #MyText (A) DYNAMIC
:
#MyText :=  ’123456’           /* extended to 6 bytes
WRITE *LENGTH(#MyText)         /* is 6
CALLNAT  ’SUB1’ USING #MyText
WRITE *LENGTH(#MyText)         /* is 8; allocated size is 8


Subpgm SUB1:
DEFINE DATA PARAMETER
    1 #MyParm (A)  DYNAMIC BY VALUE RESULT
:
WRITE *LENGTH(#MyParm)     /*is 6; temporary value space of 6 bytes is allocated for #MyParm
#MyParm := ’1234567’       /* extended to 7
#MyParm := ’12345678’      /* allocated size=8 bytes
EXPAND DYNAMIC VARIABLE #MyParam TO 10
WRITE *LENGTH(#MyParm)     /* is 8; allocated size is 10
END                        /* contents of  #Myparm is moved (back) to #MyText
                           /* used length is 8; #MyText is extended to 8
```

Example 2:

```
DEFINE DATA LOCAL
    1 #MyText (A) DYNAMIC
:
#MyText :=  ’123456’           /* extended to 6 bytes
WRITE *LENGTH(#MyText)         /* is 6
CALLNAT  ’SUB2’ USING #MyText
WRITE *LENGTH(#MyText)         /* is 8; allocated size is 10 (extended in SUB2)


Subpgm SUB2:
DEFINE DATA PARAMETER
    1 #MyParm (A)  DYNAMIC
:
WRITE *LENGTH(#MyParm)     /* is 6
#MyParm := ’1234567’       /* used length is 7
#MyParm := ’12345678’      /* extended to 8 bytes
EXPAND DYNAMIC VARIABLE #MyParm TO 10
WRITE *LENGTH(#MyParm)     /* is 8; allocated size is 10
END
```

CALL 3GL Program

Dynamic and large variables can sensibly be used with the CALL statement when the option INTERFACE4 is used. The usage of this option leads to an interface to the 3GL program with a different parameter structure. This usage requires some minor changes in the 3GL program, but provides the following significant benefits as compared with the older FINFO structure. For further information on the FINFO structure, see the Call Interface4 statement.

- no limitation on the number of passed parameters (former limit 40)
- no limitation on the parameter’s data size (former limit 64K per parameter)
- full parameter information can be passed to the 3GL program including array information.
  Exported functions are provided which allow secure access to the parameter data (formerly you had to take care not to overwrite memory inside of Natural)

Before calling a 3GL program with dynamic parameters, it is important to ensure that the necessary buffer size is allocated. This can be done explicitly with the EXPAND statement.
If an initialized buffer is required, the dynamic variable can be set to the initial value and to the necessary size by using the MOVE ALL UNTIL statement. Natural provides a set of functions that allow the 3GL program to obtain information about the dynamic parameter and to modify the length when parameter data is passed back.

Example:

```
MOVE ALL ’ ’ TO  #MyDynText1 UNTIL 10000      /* a buffer of length 10000 is allocated
                                              /* #MyDynText1 is initialized with blanks
                                              /* *LENGTH is set is set to 10000
CALL INTERFACE4  #3GLprog  USING  #MyDynText1
write *LENGTH(#MyDynText1)                    /* used length may have changed in the 3GL program
:
```

For a more detailed description refer to the CALL statement documentation in the Statements Manual.

Work File Access with Large and Dynamic Variables

Large and Dynamic Variables can be written into work files or read from work files using the two work file types PORTABLE and UNFORMATTED. For these types there are no size restrictions for large/dynamic variables.

The other work file types (ASCII, ASCII-COMPRESSED, ENTIRECONNECTION, SAG and TRANSFER) cannot handle dynamic variables and will produce an error. Large variables for these work file types pose no problem unless the maximum field/record length is exceeded (field length 255 for ENTIRECONNECTION and TRANSFER, record length 32767 for the others).

For the work file type PORTABLE, the field information is stored within the work file. The dynamic variables are resized during READ if the field size in the record is different from the current size.

The work file type UNFORMATTED can be used, for example, to read a video from a database and store it in a file directly playable by other utilities. In the WRITE WORK statement, the fields are written to the file with their byte length. All data types (DYNAMIC or not) are treated the same. No structural information is inserted. Note that Natural uses a buffering mechanism, so you can expect the data to be completely written only after a CLOSE WORK. This is especially important if the file is to be processed with another utility while Natural is running.

With the READ WORK statement, fields of fixed length are read with their whole length. If the end-of-file is reached, the rest of the current field is filled with blanks. The following fields are unchanged.
In case of DYNAMIC data types, the complete rest of the file is read unless it exceeds 1 GB. If the end of file is reached, the remaining fields (variables) are kept unchanged (normal Natural behavior).

DDM Generation and Editing for Varying Length Columns

Depending on the data types, the related database format A or format B is generated. The Natural length is not set to a defined value for a varying length column in the DBMS. In the case of varying length columns, the original data type (and the maximum length, if defined) of the DBMS will be documented in the Remark column. The keyword DYNAMIC is displayed at the length field position.

For all varying length columns, an LINDICATOR field L@<column-name> will be generated. For the databases' data type VARCHAR, an LINDICATOR field with format/length I2 will be generated. For large data types (see list below) the format/length will be I4.

In the context of database access the LINDICATOR handling offers the possibility to get the size of the field to be read or to set the size of the field to be written independent from a defined buffer length (or from *LENGTH). Usually, after a retrieval function, *LENGTH will be set to the corresponding length indicator value.

Example DDM:

```
T  L  Name                      F   Leng        S   D   Remark
   :
   1  L@ PICTURE1               I   4                                     /* length indicator
   1  PICTURE1                  B   DYNAMIC             IMAGE
   1  N@PICTURE1                I   2                                     /* NULL indicator
   1  L@TEXT1                   I   4                                     /* length indicator
   1  TEXT1                     A   DYNAMIC             TEXT
   1  N@TEXT1                   I   2                                     /* NULL indicator
   1  L@DESCRIPTION             I   2                                     /* length indicator
   1  DESCRIPTION               A   DYNAMIC             VARCHAR(1000)
   :
   :
~~~~~~~~~~~~~~~~~~~~~~Extended Attributes~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/* concerning PICTURE1
 Header              :    ---
 Edit Mask           :    ---
 Remarks             :   IMAGE
 Length              :   DYNAMIC
```

The generated formats are varying length formats. The Natural programmer has the possibility to change the definition from DYNAMIC to a fixed length definition (extended field editing) and can change, for example, the corresponding DDM field definition for VARCHAR data types to a multiple value field (old generation).

Example:

```
T  L  Name                  F   Leng         S   D   Remark
   :
   1  L@ PICTURE1           I   4                                        /* length indicator
   1  PICTURE1              B   1000000000           IMAGE
   1  N@PICTURE1            I   2                                        /* NULL indicator
   1  L@TEXT1               I   4                                        /* length indicator
   1  TEXT1                 A   5000                 TEXT
   1  N@TEXT1               I   2                                        /* NULL indicator
   1  L@DESCRIPTION         I   2                                        /* length indicator
M  1  DESCRIPTION           A   100                  VARCHAR(1000)
     :
     :
~~~~~~~~~~~~~~~~~~~~Extended Attributes~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/* concerning PICTURE1
 Header            :    ---
 Edit Mask         :    ---
 Remarks           :   IMAGE
 Length            :    ---
```

Accessing Large Database Objects

To access a database with large objects (CLOBs or BLOBs) a DDM with corresponding large alpha or binary fields is required. If a fixed length is defined and if the database large object does not fit into this field, the large object is truncated. If the programmer does not know the definitive length of the database object, it will make sense to work with dynamic fields. As many reallocations as necessary are done to hold the object. No truncation is performed.

Example Program:

```
DEFINE DATA LOCAL
:
1 person VIEW OF xyz-person
  2 nachname
  2 vorname_1
  2 L@PICTURE1                   /* I4 length indicator for PICTURE1
  2 PICTURE1                     /* defined as dynamic in the DDM
  2 TEXT1                        /* defined as non-dynamic in the DDM
:
END-DEFINE
:
SELECT * INTO VIEW person FROM xyz-person                   /* PICTURE1 will be read completely
                             WHERE nachname = ’SMITH’       /* TEXT1 will be truncated to fixed length 5000
:
   WRITE  ’length of PICTURE1: ’  L@PICTURE1                /* the L-INDICATOR will contain the length
   :                                                        /* of PICTURE1 (= *LENGTH(PICTURE1)
   /* do something with PICTURE1 and TEXT1
:
   L@PICTURE1  :=  100000
   INSERT INTO xyz-person  (*) VALUES (VIEW person)         /* only the first 100000 Bytes of PICTURE1
   :                                                        /* are inserted
:
END-SELECT
```

If a format-length definition is omitted in the view, these are taken from the DDM.
In reporting mode; it is now possible to specify any length, if the corresponding DDM field is defined as DYNAMIC. The dynamic field will be mapped to a field with a fixed buffer length. The other way round is not possible.

DDM **format/length definition**	**VIEW** **format / length definition**	
(An)	-	valid
	(An)	valid
	(Am)	only valid in reporting mode
	(A) DYNAMIC	invalid
(A) DYNAMIC	-	valid
	(A) DYNAMIC	valid
	(An)	only valid in reporting mode
	(Am / i : j)	only valid in reporting mode

(equivalent for format B variables)

Parameter with LINDICATOR Clause in SQL Statements

If the LINDICATOR field is defined as I2 field, the SQL data type VARCHAR is used for sending or receiving the corresponding column. IF the LINDICATOR host variable is specified as I4, a large object data type (CLOB/BLOB) is used.

If the field is defined as DYNAMIC, the column is read in an internal loop up to its real length. The LINDICATOR field and *LENGTH are set to this length. In case of fixed length field, the column is read up to the defined length. In both cases, the field is written up to the value defined in the LINDICATOR field.

Performance Aspects with Dynamic Variables

EXPAND and REDUCE

The amount of the allocated memory of a dynamic variable may be reduced to a specified size using the REDUCE DYNAMIC VARIABLE statement. In order to (re)allocate a variable to a specified size, the EXPAND statement can be used.
(If the variable should be initialized use the MOVE ALL UNTIL statement.)

Example:

```
DEFINE DATA LOCAL
:
#MyDynText1    (A)  DYNAMIC
# len          (I4)
:
END-DEFINE

#MyDynText1 := ’a’                  /* used length is 1, value is ’a’; allocated size is still 1

EXPAND DYNAMIC VARIABLE  #MyDynText1 TO 100
                                    /* used length is still 1, value is ’a’; allocated size is 100

CALLNAT #subprog  USING  #MyDynText1
write *LENGTH(#MyDynText1)          /* used length and allocated size may have changed in the subprogram

#len := *LENGTH(#MyDynText1)
REDUCE DYNAMIC VARIABLE  #MyDynText1  TO  #len
                                    /* if allocated size is greater than used length, the unused memory is released
:
REDUCE DYNAMIC VARIABLE  #MyDynText1  TO  0
                                    /* free allocated memory for dynamic variable
END
```

Rules:

- Use dynamic operands where it makes sense.
- Use EXPAND if upper limit of memory usage is known.
- Use REDUCE if the dynamic operand will no longer be needed.

Introduction to Event-Driven Programming

The following topics are covered below:

- What is an Event-Driven Application?
- GUI Development Environments
- GUI Design Tips
- Tasks Involved in Creating an Application
- Tutorial - Overview
- Creating a Dialog
- Assigning Attributes to the Dialog
- Creating Dialog Elements Inside the Dialog
- Assigning Attributes to the Dialog Elements
- Creating the Application's Local Data Area
- Attaching Event Handler Code to the Dialog Element
- Checking, Stowing and Running the Application
- Basic Terminology

For further information on Event-driven Programming see Event-Driven Programming Techniques.

What is an Event-Driven Application?

- Program-Driven Applications
- Event-Driven Applications
- What is Happening Here?
- Writing Event-Driven Code
- Components of an Event-Driven Application

Event-driven applications represent a new approach to development in addition to the program-driven approach. Natural offers you both. Event-driven programming allows the application to be driven by input received through the graphical user interface.

In program-driven applications, the application controls the portions of code that execute - not an event. Execution starts with the first line of executable code and follows a defined pathway through the application, calling additional programs as instructed in the predetermined sequence.

In event-driven programming, the user's action or a system event triggers the code attached to that event. Thus, the order in which your code executes depends on which events occur, which in turn depends on what the user does. This is the essence of graphical user interfaces and event-driven programming: The user is in charge, and the code responds. Even though event-driven programming is possible in character-oriented interfaces, it is more common in graphical user interfaces.

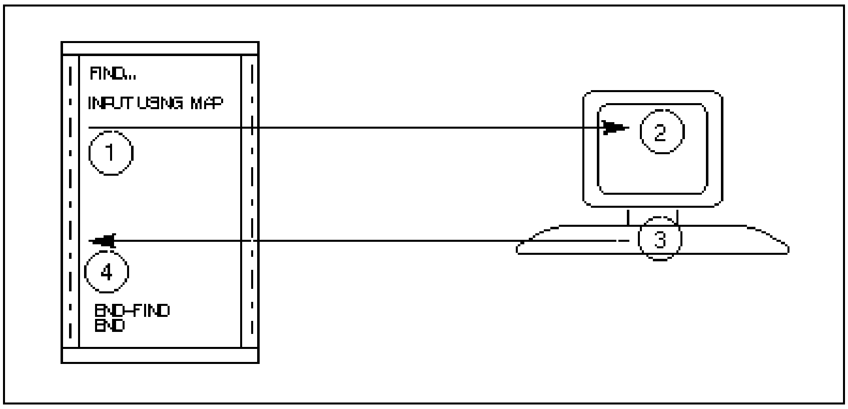
Because you cannot predict what the user will do, your code must make a few assumptions when it executes. For example, the application might assume that the user added text to an edit-area control before pressing the OK button.

When you must make assumptions, you should try to structure your application so these assumptions are always valid. For example, to ensure the user added text, you can disable the button and enable it only when the change event occurs for the edit-area control.

Your code can trigger additional events as it performs certain operations. For example, moving the slider in a scroll-bar control triggers the change event.

The following diagrams illustrate the difference between program-driven and event-driven applications.

Program-Driven Applications

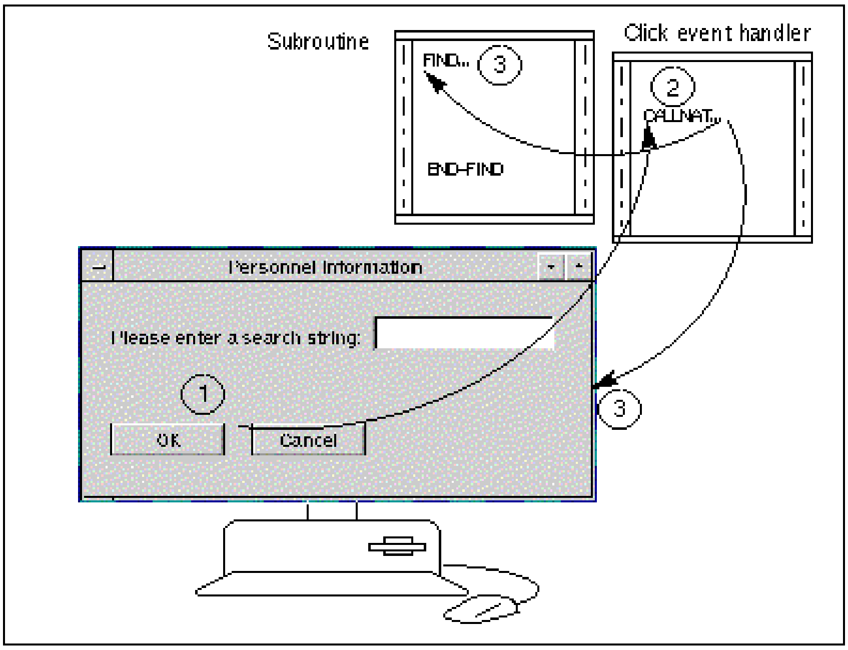


In typical program-driven applications, the following sequence of steps applies:

1. The program sends a screen to the terminal.
2. The user reacts by filling in the data fields.
3. The user then presses ENTER or a function key.
4. The program then decides whether or not the user’s entries are valid.

If the data are valid, it processes the results until it reaches an END statement.

Event-Driven Applications



In typical event-driven applications, the following sequence of steps applies:

1. The user requests an action on the screen.
2. The event handler code reacts in the background according to the context.
3. If certain conditions are fulfilled, the executed event handler code triggers other Natural code (here: a subroutine) or returns control to the screen.

In the program-driven approach, the user interacts with the code through the ENTER and function keys, the user of an event-driven application triggers specific pieces of code (event handlers). Typically, an event-driven application is not executing any code when waiting for user input; in the same situation, the program-driven application might be processing an INPUT statement.

What is Happening Here?

Graphical user interface programs require you to write programs that react to isolated events initiated by the user.

An event is an action recognized by a dialog or a dialog element. Event-driven applications execute code in response to an event. Each dialog or dialog element has a predefined set of events. If one of these events occurs, Natural invokes the code in the associated event handler.

You decide if and how the dialogs and dialog elements in your application respond to a particular event. When you want a program to respond to an event, you write event code for that event.

Writing Event-Driven Code

For each dialog or dialog element you create, Natural predefines a set of events to which your program (event handler) can respond. It is easy to respond to events: dialogs and dialog elements have the built-in ability to recognize user actions and execute the code associated with them.

You do not have to write code for all events. When you do want a dialog object to respond to an event, you write event code that Natural executes in response to that event.

In a typical event-driven application, the following series of actions takes place:

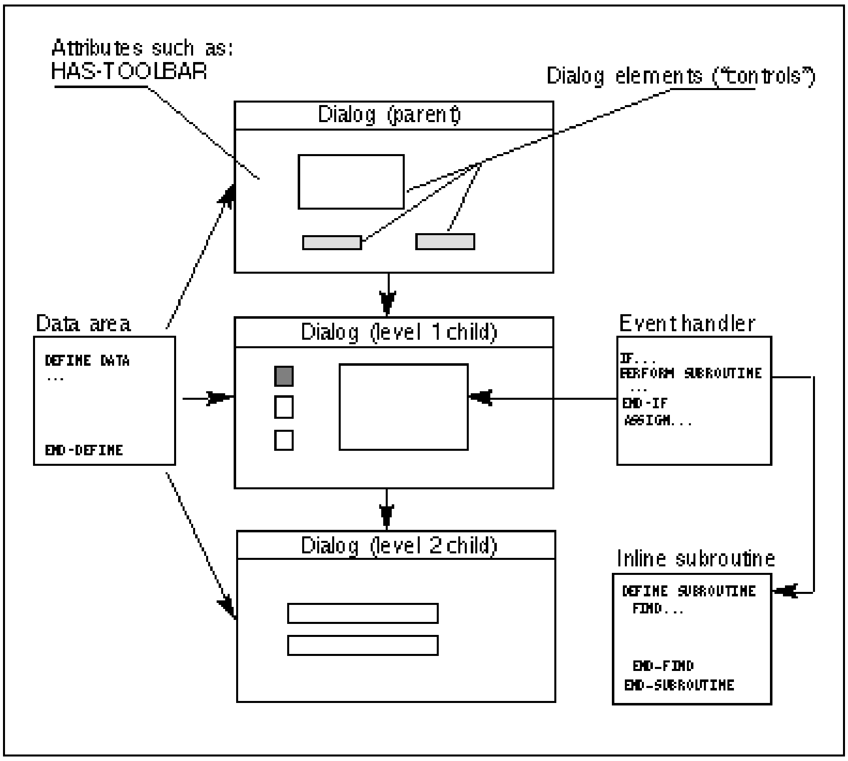
- A dialog or dialog element recognizes an action as an event. The action can be caused by the user (such as a click or keystroke).
- If there is event code corresponding to the event, it is executed.
- The application waits for the next event.

The event code you write to respond to events can perform calculations, get input, and manipulate parts of the interface. Using Natural, you manipulate dialogs or dialog elements by changing the values of their attribute settings.

Note:
Avoid creating cascading events in your code caused by events occurring repeatedly. For example, when the user drags the slider in the scroll-bar control, the current SLIDER attribute setting is automatically changed and the change event is triggered. If the code attached to the change event also changes the current SLIDER attribute setting, then the change event is triggered again, the current SLIDER attribute setting is again adjusted, the change event is once again triggered, and so on. At this rate, you quickly run out of memory.

Components of an Event-Driven Application

Overview



Dialogs

The dialog is the central Natural object in an event-driven application. An event-driven application is started by running or executing the base dialog. This may open other dependent dialogs when the OPEN DIALOG statement is specified. As opposed to program-driven applications, these dialogs are usually modeless, that is, all open dialogs can be processed concurrently by the end user. The application terminates when the base dialog is closed.

You create a dialog with the dialog editor. Just like the map editor, the dialog editor assembles a Natural object from the specification of the dialog window and its dialog elements, the global data area (GDA), the local data areas (LDAs), the parameter data areas (PDAs), the subroutines and the specified event handler sections.

At runtime of the dialog, there is a difference between the runtime instance identified by the system variable *DIALOG-ID and the GUI instance (handle) of the dialog window (the default handle name is #DLG$WINDOW).

Whenever you want to work with more than one dialog in your application, you must decide how the base dialog window relates to the other dialogs. First you have to decide whether the application should be MDI (Multiple Document Interface) or not.

If you have opted for an MDI application, the base dialog must be of the type "MDI frame window" and the dependent dialogs must be of the type "MDI child window" and "Standard window".

If you have opted for non-MDI, the application may contain only dialogs of the type "Standard window".

Dialogs of type "Standard window" can have the styles Popup, Modal or Dialog Box.

Dialog Elements

Almost all dialog elements are graphical elements inside a dialog that allow the end user to interact with the event-driven application. After a dialog has been opened with the dialog editor and its attributes have been set (see below), the programmer will go on to "draw" the dialog elements inside the window; usually, this comprises a menu control, possibly a toolbar, and other elements, such as push-button controls, input-field controls.

"Drawing" a dialog element means that you select the type of dialog element from the dialog editor’s menu or toolbar, and use the mouse to place it at the desired location. It is also possible to define a grid where the dialog elements can be placed more conveniently by aligning them to the grid.

Attributes

Attributes are the properties of dialogs and dialog elements. After creating a dialog or dialog element, you double-click with the mouse on it and the window with the corresponding attributes appears. You can then set the attributes to a value; if not, they remain at the system default value. The attributes window also contains a push-button control that opens up the event handler window.

Event Handlers

The event handlers represent the Natural code that is triggered when an event occurs. A click event occurs, for example, when the end user clicks on a push-button control. Inside the event handler window, you must first select the type of event from the list of events available for the dialog or dialog element (the one whose attributes have just been set). Then, the code window is enabled and Natural code can be entered.

Data Areas - Global, Local, Parameter

- A global data area (GDA) is used to share data fields between Natural objects within the application. One GDA per application may be specified.
- A local data area (LDA) contains the data fields private to the dialog.
- A parameter data area (PDA) is always present in dialogs. It is used to pass parameters to a dialog in the OPEN DIALOG or SEND EVENT statements. In these statements, parameters are passed either by specifying their name (WITH clause), or by listing parameters one after the other. You can use the dialog editor PDA window to type in your PDA in free-form style or to include PDAs defined externally.

Inline Subroutines

An inline subroutine defines standard code to be used for a frequently needed task called by a number of event handlers. You access an inline subroutine window via the "Inline Subroutines" push-button control in the dialog window.

GUI Development Environments

To understand the functions of Natural, you must first understand the environment in which it runs.

A graphical user interface (GUI) environment differs from a traditional mainframe environment in at least two important ways:

- Applications share screen space. A Natural application runs in a group of one or more windows and rarely occupies the full screen.
- Applications share computing time. An application cannot run continually, or if it does, it must run in the background.

Using Natural, your applications share computing time and other resources (such as the clipboard). An event-driven application consists of dialogs and dialog elements that wait for a particular event to happen.

While your application is waiting to execute an event, it remains on the desktop (unless the user closes the application). In the meantime, the user can run other applications, resize windows, or customize system settings (such as color). However, your code is always present, ready to be activated when the user returns to your application.

GUI Design Tips

- Do Your Research
- Screen Design
- Menu Design
- Color Usage
- Consistency Check

Designing the screens for a GUI application requires different knowledge than designing the 3270 screens for a mainframe. Why is it different?

It is different, because GUI applications put the users in control; these applications are non-modal and unstructured. The users choose the order in which they access windows, and fields within the windows. Traditional database applications often require the users to perform operations in a specific order; these applications are form-oriented and structured.

Designing a GUI screen is also different, because the GUI interface has different capabilities than a traditional mainframe interface. You can design windows that incorporate dialog elements, such as push-button controls and list-box controls. As you design your GUI windows, which are called dialogs in event-driven Natural, you define the font type and size of the text, the background and foreground colors, and the size of each window.

The following sections provide some tips for effective GUI design.

Do Your Research

- Spend a few hours with your users before prototyping.

A couple of sessions with your users to iron out their needs, likes, and dislikes is enough to give you to a good basis for beginning your design.

- Take some ideas from existing GUI designs.
  Save time by not re-inventing the GUI. Try out other GUIs with an eye for what works and what does not. Consistency within GUIs helps users learn to use new applications, improves efficiency, and reduces training costs. Get user feedback on existing GUI applications - listen to their likes and dislikes rather than develop a prototype that replicates the weaknesses of poor GUI design.

- Develop your ideas on paper before spending time developing the application online.
  It is faster for you to run through a number of screen design options for your main windows on paper before spending time to create multiple prototypes online. It is quicker than coding and you do not become attached to poor designs.

If you include your users in the development process, they can quickly comment about their needs and likes before the application is installed in the system. Try to use a paper prototype before reaching for the online development tool.

Screen Design

- Design multiple windows for related subject matter.
  Unlike designing for 3270 monitors, where you try to maximize the number of fields per screen, GUI screens are better designed using subwindows. You can, for example, have the essential fields in the main window, and all optional or supplemental information stored in one or more subwindows. Subwindows can include choices, such as drop-down lists, for the user to browse through if they do not know the information to input into the main window. Messages and field-dependent information are more effectively presented in supplemental windows than in the main window.

- Design clear, uncluttered windows.
  Avoid cluttering your windows with more than three colors, multiple graphics, and a variety of shapes. Balance your objects on the screen with lots of white space so users are not overwhelmed by variety and distracted by the presentation. Try to keep shapes and objects to a minimum and the number of colors low.

- Design accessible, not overwhelming, windows.
  Multiple fonts, font sizes, font types or families, and color schemes can overwhelm your users, making your application seem inaccessible to them. Use a maximum of three fonts, font sizes, and font types per window. Avoid using italics and serif fonts because they often break up on the screen. Use color sparingly. Neutral colors are kindest to your users eyes. Though vibrant reds and greens are very eye-catching, remember that your users spend a lot of their day working in the windows you design.

- Design for both keyboard and mouse use.
  Some users prefer using the keyboard and memorize the short cut commands, while other users are more comfortable using the mouse. Each action should be accessible by both the mouse and the keyboard.

- Design the windows according to your users’ needs.
  Though it is tempting to create fabulous-looking screens with lots of functionality, if your users do not use it, it is of no value. Remember that you are designing the application for your users to get a job done, not for you to experiment with all the functionality you have available. First find out what your users need, then tailor your design to meet their needs. You design screens with different purposes in different ways. If you want to prompt the user, you use a conversational style; if you want the user to enter values from a form, you use a data-entry style.

Conversational Screens

- Design conversational screens with field prompts.
  In a conversational-based style, users enter data from a conversation (travel reservations, for example). Conversational-based styles, in which the user relies on the screen for prompting, can be rich with labels, hints, instructions, and even questions for the users to ask their clients.

Data-Entry Screens

- Design data-entry screens with terse labels.
  In a form-based style, users enter data from a form. Each line on the input screen must match a line on the form - and the lines must be in the same order. To maintain a line-for-line correspondence, you can abbreviate labels. Headings and instructions are kept to a minimum. The only purpose of labels is to help users find their places again after interruptions.

Menu Design

The following three criteria are recommended for designing menus.

- Organize menus using the conventions defined by the operating system on which your users run the application.
  Microsoft Windows, for example, recommends certain menus ("File", "Edit", and "View", for example), options on menus ("Cut", "Copy", and "Paste" on the "Edit" menu, for example), and a particular order of the menus on the menu bar (Help always appears at the right margin, for example).

- Arrange menus by frequency of use and decide this information through observation or usability testing.
  Anticipate whether usage changes as users become more expert. Watch that this does not violate conventions established for the operating system.

- List menu items alphabetically.
  Remember to follow the operating-system conventions and user recommendations for frequency of using menu items.

Color Usage

- Be as conservative as possible with color.
  Humans can remember the meaning of no more than five colors at a time, plus or minus two.

- Use color as an additional signal, not as the primary signal.
  Using bright red text to warn a user is not enough; add a warning tone. Eight percent of all males are red-green color-blind and may not notice the red text.

- On charts, do not use colors without adding a secondary key (for example, a broken or solid underline).
  Users with black and white monitors must be able to understand the key without the benefit of color. Also, most users do not have color printers.

Consistency Check

- Be consistent throughout the application.
  Do not change fonts, colors, or shapes for related subjects. For example, design all the OK buttons in an application with the same shape, size, color, and font. If related objects are presented in different ways, users cannot use the visual clues, taking them longer to become comfortable with the application. Present similar actions in a similar way, using the same font, color, and size for related buttons.

- Adopt a naming convention (and stick with it throughout the application).
  While traditional programs tend to have one large program you modify for a name change, object-oriented programs have numerous pieces of event code that you must edit individually every time you make a name change. When you design GUI applications, you must be much more rigorous about sticking to naming conventions. This avoids a lot of cleaning up time later.

Tasks Involved in Creating an Application

There are a number of main tasks you perform to create an application in event-driven Natural. The order in which they are explained in this section is the typical order in which you perform them. However, this sequence is not inflexible. For example, you may very well test a dialog several times in the process of designing it, and you will no doubt save your work more often during the development process.

- Decide whether your application is Multiple Document Interface or Single Document Interface.
- Create one or more dialogs.
- Set the attributes of the dialog(s).
- Create and place dialog elements in the dialog(s).
- Set the attributes of the dialog elements.
- Define the tab order in each of the dialogs (from the menu, choose "Dialog > Control Sequence").
- Save the dialog(s) to a name.
- Define the global data area.
- Define the local data area(s).
- Write event handler code for the dialog(s).
- Write inline subroutines for the dialog(s).
- Write event handler code for the dialog elements.
- Stow the dialog(s).
- Test (check and run) the dialog(s).
- Execute the application.

The following short tutorial introduces you to the most frequently performed tasks.

Tutorial - Overview

This section is a simple tutorial that demonstrates how to add the components of an event-driven application one after the other. The tutorial describes how to develop a small sample application consisting of one dialog. The application you will create is a degressive depreciation calculator.

You can use this calculator, for example, to find out the value of your car by entering how much the car was worth when you bought it, how many years you have owned it, and the percentage by which the value of the car decreases each year.

You can save your application at any stage, allowing you to interrupt the tutorial and continue at a later time where you left.

▶ To develop the sample application

1. Create a new dialog (represented by a window).
2. Assign the attributes to your dialog (decide the window’s settings).
3. Create the dialog elements in the dialog (decide how the user can interact).
4. Assign the attributes to your dialog elements (decide attribute settings).
5. Create the application’s local data area (define the variables that allow the event handler to use the end user’s numeric input).
6. Attach event handler code to the dialog element (decide what happens at runtime when the user interacts).
7. Check, stow and run the application.

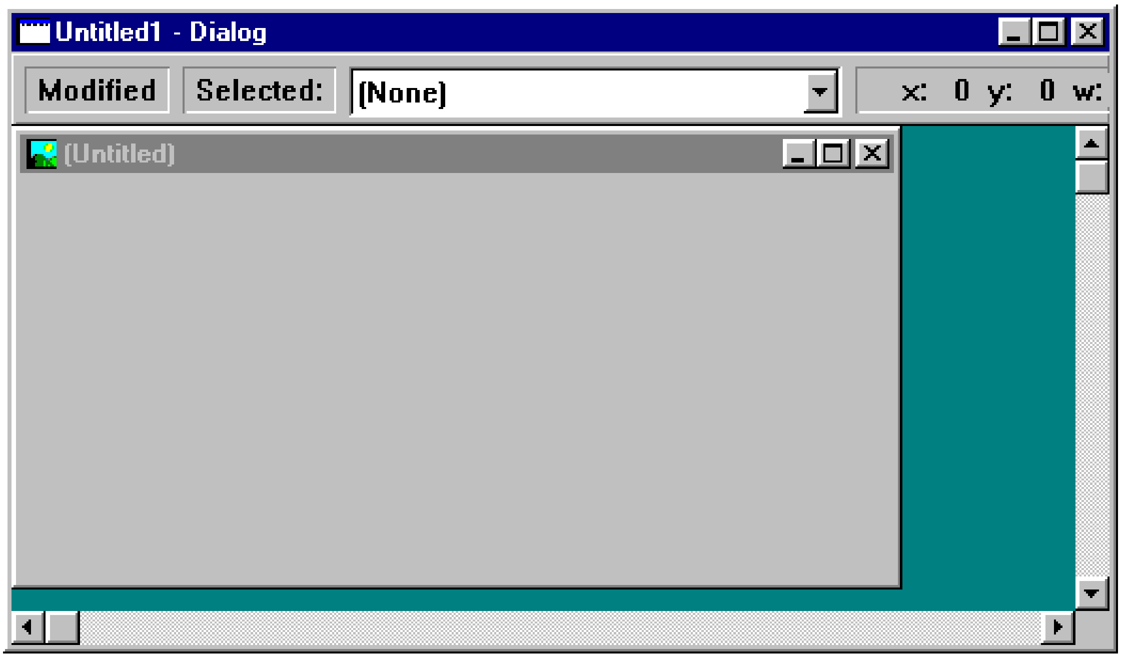
Apart from creating the local data area, this is the minimal number of steps required to create any event-driven application.

Creating a Dialog

To create a new Dialog

1. Invoke Natural.
2. From the Natural menu, select "Object > New > Dialog".

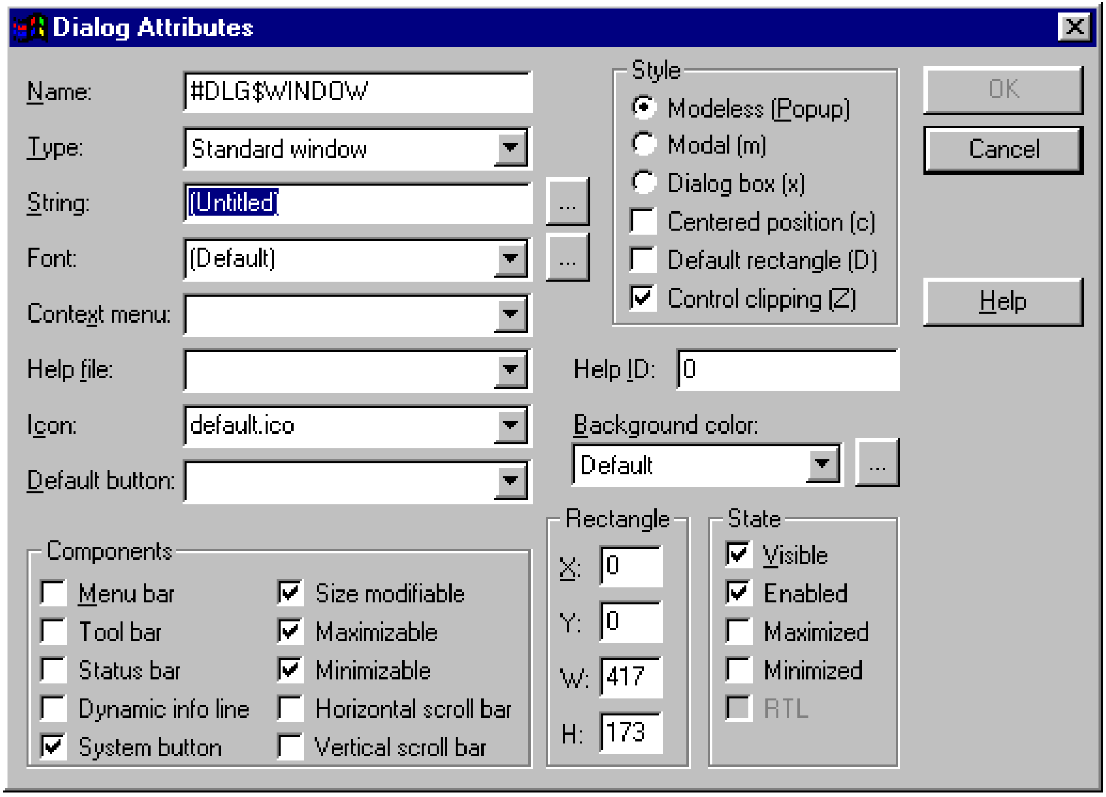
The Natural window displays the dialog editor’s menu bar and toolbar. It displays an editing window called "Untitled1-Dialog". You can resize this editing window. The editing window contains the new dialog window, titled "(Untitled)". You can also resize this new dialog window, or use the editing window’s scroll bars.



Assigning Attributes to the Dialog

To assign attributes to the dialog

1. Double-click inside the dialog window.
   The "Dialog Attributes" window appears.



Dialog Attributes

Name: #DLG$WINDOW
Type: Standard window
String: (Untitled)
Font: (Default)
Context menu:
Help file:
Icon: default.ico
Default button:

Style
● Modeless (Popup)
○ Modal (m)
○ Dialog box (x)
□ Centered position (c)
□ Default rectangle (D)
☑ Control clipping (Z)

Help ID: 0
Background color: Default

Components
□ Menu bar
□ Tool bar
□ Status bar
□ Dynamic info line
☑ System button
☑ Size modifiable
☑ Maximizable
☑ Minimizable
□ Horizontal scroll bar
□ Vertical scroll bar

Rectangle
X: 0
Y: 0
W: 417
H: 173

State
☑ Visible
☑ Enabled
□ Maximized
□ Minimized
□ RTL

2. With the cursor in the "String" field, type in the new dialog window’s title: "Degressive Depreciation".
3. Open the "Background" selection box by clicking on the down arrow.
   A list with predefined colors drops down.
4. Mark the desired color, for example "Gray".
5. Choose OK.

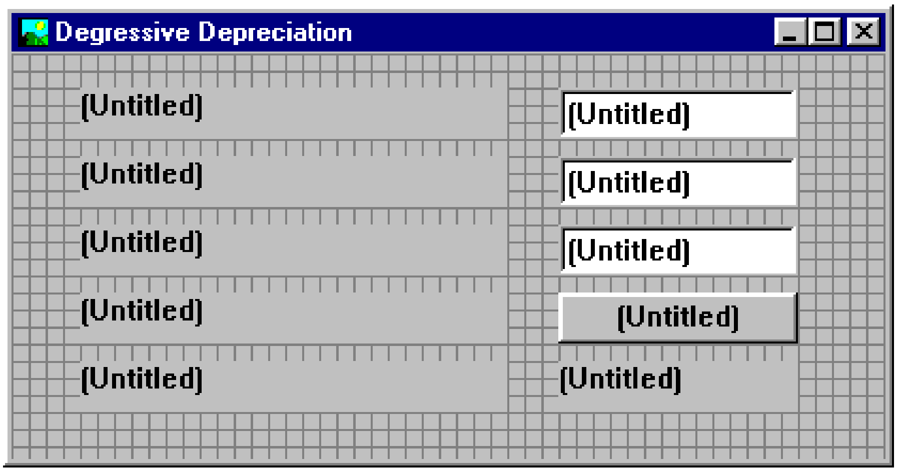
The attributes window closes.
You have set the attribute STRING to the value "Degressive Depreciation" and the attribute BACKGROUND-COLOUR-NAME to the value of your desired color, for example GRAY.

Creating Dialog Elements Inside the Dialog

To create the dialog elements inside the dialog

1. From the menu bar, select "Tools > Options...".
   The "Options" dialog box appears. Choose the "Dialog Editor" tab.
2. Choose "Lines".
   This decides the way your grid will be displayed.
3. Choose OK to confirm the change.
   The grid now helps you position and align the dialog elements.
4. To display the grid from the "Options" dialog, check the ’display grid’ option on the "Dialog Editor > tab and choose OK to confirm the change.
5. From the menu bar, select "Insert > Text constant", or click on the toolbar button representing a text-constant control.
6. Move the cursor to the upper left corner of the dialog window.
   Ensure that the Editor window’s status bar displays an x and a y value of less than 50. Note that at this time, the text-constant control’s width and height has an undefined value.
7. Click to fix the text-constant control’s position.
   A grey rectangle representing the dialog element appears, surrounded by small black squares. At the same time, the status bar indicates that "#TC-1" is selected.
8. Point to one of the small black squares.
   The cursor shape now indicates the direction in which you can resize the text-constant control.
9. Resize "#TC-1" to a width of about 200.
10. From the menu bar, select "Edit > Copy" followed by "Edit >Paste" and a new text-constant control "#TC-2" is created on top of "#TC-1". Move the new text-constant control to a position below the first one by clicking and dragging via the mouse or via the keyboard arrow keys with the <SHIFT> held down.
11. Create another three text-constant controls below (in the same way).
12. Create three input-field controls in the upper right corner of the dialog window (by creating the first and duplicating it, as above). These input-field controls should have a height of 36. Align them horizontally with respect to each other and vertically with respect to the upper three text-constant controls.
13. Create a push-button control below the three input-field controls and align it.
14. Create a text-constant control below the push-button control and align it.
15. Align the whole layout until you get a harmonious, well-balanced picture.

Your dialog could now look like this:

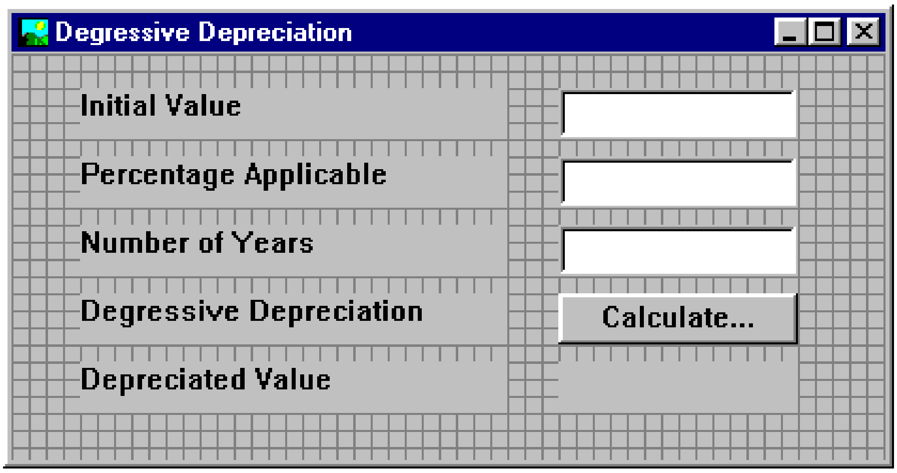


Assigning Attributes to the Dialog Elements

To assign attributes to the dialog elements

1. Double-click on the text-constant control "#TC-1".
   The corresponding attributes window appears.
2. In the "String" entry, type in the text string to be displayed: Initial value.
3. Choose OK or press ENTER.
   The attributes window closes.
4. Set the following text strings for the four text-constant controls below: Percentage Applicable, Number of Years, Degressive Depreciation, Depreciated Value.
5. From the three input-field controls, remove any text string
6. Set the following text string for the push-button control: "Calculate..."
7. From the last text-constant control, remove any text string.
   It will be used for output purposes.

Your dialog should now look like this:



Creating the Application’s Local Data Area

The local data area in this application defines the application’s linked variables. These linked variables receive the numeric values that the end user has entered in the input-field controls. The variables and their values are used in the calculation of the push-button control’s click event handler code.

To prepare the creation of your local data area, your input-field controls must use linked variables

1. Double-click on the first input-field control "#IF-1".
   The corresponding attributes window appears.
2. Click on the "..." push-button to the right of the "String" entry.
   The "Source for #IF-1.STRING" dialog box appears.
3. In the "Attribute Source" group frame, click (and enable) the "Linked variable" radio button
4. In the "Variable Name" entry, enter: #INITIAL-VALUE.
5. Choose OK twice to leave both the "Source for #IF-1.STRING" dialog box and the attributes window.
6. Set the following linked variable names for the remaining two input-field controls: #PERC-APPLIC, #YEAR-NUM.

To create the application’s local data area

1. From the menu bar, select "Dialog > Local data area...".
   The "Dialog Local Data Area" definition section appears.
2. Define your local data as follows:
   1 #INITIAL-VALUE (N6.2)
   1 #PERC-APPLIC (N2.1)
   1 #YEAR-NUM (N2)
3. Choose OK.

Natural will now be able to process the input data.

Attaching Event Handler Code to the Dialog Element

To attach event handler code

1. Select the push-button control labelled "Calculate...".
2. From the menu bar, select "Control > Event Handlers..." The corresponding Event handler definition section appears.
   The "Click" Event is preselected: when the end user clicks on this push-button control, the specified Natural code will be triggered.
3. In the event handler editing area, enter the following Natural code in free form:
   ```
   #RESULT:= #INITIAL-VALUE * ( ( ( 100 - #PERC-APPLIC )
   / 100 ) ** #YEAR-NUM )
   MOVE EDITED #RESULT (EM=Z(5)9.99) TO #TC-6.STRING
   ```
4. Choose OK to close the editing area, and choose OK again to close the attributes window.

Checking, Stowing and Running the Application

To check the application for syntax errors

1. From the menu bar, select "Object > Check".
   A dialog box comes up with a Natural error: a variable needs to be declared.
2. In the dialog box, select the "Edit" push button.
   The dialog’s code is displayed, the cursor pointing to the error.
3. Select "Cancel".
4. Select "Dialog > Local data area"
5. Add the definition "1 #RESULT (N6.2)".
6. Select OK.
7. Check your application again.

The Information message box should now confirm that the check was successful.

To stow your application

1. From the menu bar, select "Object > Stow".
   The "Stow Dialog As" dialog box appears.
2. Enter the name "Degrdep".
3. From the "Libraries" list box, select the library where you want the dialog to be stowed.
4. Choose OK.

The Information message box now confirms that the dialog was stowed successfully.

To test your application

From the menu bar, select "Object > Run".

Basic Terminology

Event-driven Natural uses the following basic terminology:

Attribute

A property of a dialog or a dialog element which can assume specific values. Example: If the HAS-STATUS-BAR attribute is set to TRUE for a dialog, then the dialog contains a status bar. The following operations may be made on attributes:

Operation	Result
Query	In event handler code, you can query an attribute’s value at runtime. Example: **#L:= #DLG$WINDOW.HAS-STATUS-BAR**
Set	In event handler code, you can set an attribute to a value in the global attribute list before you create a dialog element dynamically. Example: **#PUSH.STYLE:= ’O’** **PROCESS GUI ACTION ADD WITH #W PUSHBUTTON #PUSH**
Modify	In event-handler code, you can modify an attribute value of an existing dialog element at runtime. Example: **#PUSH.STYLE:= ’C’**

Base Dialog

This is the main dialog of an application. It is started from the command line or via the object list. When this dialog is closed, all other dialogs of the application are closed as well.

Control

A type of dialog element. Examples: edit-area control, push-button control, list-box control.

Dialog

A Natural object similar to a map or a program that represents a window in an event-driven application, plus all event handlers and attributes directly attached to the window. It can be a window, a modal window, a dialog box, an MDI child window, and an MDI frame window. The window as such is identified by its handle, the whole dialog is represented by the value of the system variable *DIALOG-ID.

Dialog Box

A special kind of dialog that is exclusively processed in an application. While this dialog is active, all other dialogs of the application are disabled and do not accept any user input. If a dialog invokes a dialog box with an OPEN DIALOG statement, the dialog returns from the OPEN DIALOG statement only after the dialog box is closed. This allows the application to return parameters from the dialog box to the dialog.

Dialog Editor

The Natural editor with which you create and maintain dialogs.

Dialog Element

Dialog elements are (in most cases) graphical elements inside a window that enable the end user to interact with the event-driven application. After a dialog has been created, and its attributes have been set, the programmer places the dialog elements inside the window; usually, this comprises a menu control, possibly a toolbar, and other elements, such as push-button controls and input-field controls. There are two types of elements: controls and items.

Event

Occurs when a user interacts with a dialog element. An event may also be sent from within a piece of code (user-defined event). Example: a click event occurs when the user mouse-clicks on a push-button control for which a piece of click event handler code has been specified. The system variable *EVENT contains the event name.

Event Handler

Programming code that is connected with a dialog element, and is triggered when a particular type of event occurs.

Handle

Identifies a dialog element in code and is stored in handle variables. Example: #PB-1.

Item

A type of dialog element that is part of a control. Example: selection-box item, which is part of a selection-box control.

MDI - Multiple Document Interface

Allows an application to manage several different documents or several views of the same document within the main application window (MDI frame window). These views or documents are displayed in separate MDI child windows.

MDI Child Window

Displays a view of a document within the MDI frame window of an MDI application.

MDI Frame Window

The parent window to all other child (document) windows in an MDI application.

Modal Window

Similar to a dialog box, except that if a dialog invokes a modal window with an OPEN DIALOG statement, the dialog returns from the OPEN DIALOG statement immediately after the modal window has completed opening.

SDI - Single Document Interface

As opposed to MDI applications, SDI applications do not have an MDI frame window that contains the document windows. Only a single view of a single document is displayed.

Popup

A dialog with style "Popup" is modeless and can be moved anywhere on the desktop.

Window

The basic type of window.

Event-Driven Programming Techniques

This chapter addresses the more experienced GUI programmer and describes essential programming techniques. There are two ways to program in the dialog editor:

- Use the dialog editor's menu bar and toolbar to create new dialogs or dialog elements and use the attributes window to assign attribute values to them. The dialog editor will internally generate the corresponding Natural code.
- Open an event-handler section or an inline-subroutine section and specify Natural code explicitly. This code will be added to the code that is generated internally. You can also enter parameter data areas, global data areas and local data areas in the corresponding definition sections.

You can view the current dialog's generated and specified code by choosing "Object > List" in the dialog editor's menu bar.

If you want a hands-on demonstration of how to program with the dialog editor, refer to the SYSEXEVT library. This library contains sample dialogs demonstrating basic functionality. Before accessing the sample dialogs, read the README file. Then execute the MENU dialog.

Notes:
Code written in the dialog editor must be in structured mode.

If you want to execute a Natural application using dialogs, you must use a dialog to start this application.

The following topics are covered below:

- How To Open and Close Dialogs
- How To Edit a Dialog's Enhanced Source Code
- How Dialogs, Controls and Items Are Related Hierarchically
- How To Define Dialog Elements
- How To Manipulate Dialog Elements
- How To Create and Delete Dialog Elements Dynamically
- How To Enable and Disable Dialog Elements
- Defining and Using Context Menus
- System Variables
- Generated Variables
- Message Files and Variables as Sources of Attribute Values
- Triggering User-Defined Events
- Suppressing Events
- Menu Structures, Toolbars and the MDI
- Executing Standardized Procedures
- Linking Dialog Elements to Natural Variables
- Validating Input in a Dialog Element
- Storing and Retrieving Client Data for a Dialog Element
- Creating Dialog Elements on a Canvas Control
- Working with ActiveX Controls
- Working with Arrays of Dialog Elements
- Working with Control Boxes
- Working with Error Events
- Working with a Group of Radio-Button Controls
- Working with List-Box Controls and Selection-Box Controls
- Working with Nested Controls
- Working with a Dynamic Information Line
- Working with a Status Bar

- Working with Status Bar Controls
- Working with Dynamic Information Line and Status Bar
- Adding a Maximize/Minimize/System Button
- Defining Color
- Adding Text in a Certain Font
- Adding Online Help
- Defining Mnemonic and Accelerator Keys
- Dynamic Data Exchange - DDE
- Object Linking and Embedding - OLE

For further information on Event-driven Programming see Introduction to Event-Driven Programming.

How To Open and Close Dialogs

Opening a Dialog

An event-driven application is started by executing the base dialog. Events triggered by the end user will then typically cause other dialogs to be started. The application ends when the base dialog is closed.

To open a dialog from anywhere within an event-driven application

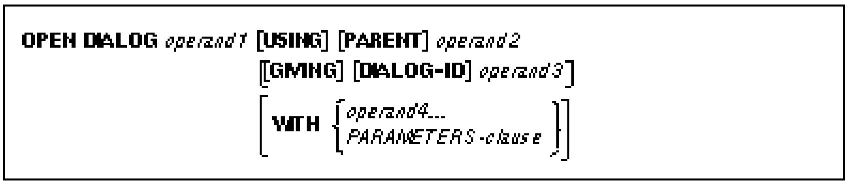
Use the statement OPEN DIALOG.

This statement causes the dialog to be loaded and the processing on its opening to be performed.

Control over processing returns from the opened dialog except for dialogs with the style "Dialog Box". For those dialog styles, control returns only after the dialog has ended.

The parameters passed are accessible only during the processing on the opening of a dialog (before-open and after-open events), except for when the parameters are declared as BY VALUE in the parameter data area of the opened dialog or when the dialog has the style "Dialog Box".

To open a dialog from anywhere within an event-driven Natural application, the following syntax is used:



```
OPEN DIALOG operand1 [USING] [PARENT] operand2
                     [[GIVING] [DIALOG-ID] operand3]
                     [WITH {operand4...         }]
                     [     {PARAMETERS-clause   }]
```

Operands

Operand1 is the name of the dialog to be opened. If the *PARAMETERS-clause* is used, *operand1* must be a constant (the name of a cataloged dialog).

Operand2 is the handle name of the parent.

Operand3 is a unique dialog ID returned from the creation of the dialog. It must be defined with format/length I4.

Passing Parameters to the Dialog

When a dialog is opened, parameters may be passed to this dialog.

As *operand4* you specify the parameters that are passed to the dialog.

With the *PARAMETERS-clause*, parameters may be passed selectively:

```
PARAMETERS {parameter-name =operand4 }_ END-PARAMETERS
```

Note: You may only use the PARAMETERS-clause if operand1 is an alphanumeric constant and if the dialog is cataloged.

Parameter-name is the name of the parameter as defined in the parameter data area section of the dialog.

To avoid format/length conflicts between operands and parameters passed, see the BY VALUE option of the DEFINE DATA statement in the Natural Statements Manual.

When passing parameters only with *operand4*, a dialog may be opened as follows:

Example:

```
/* The following parameters are defined in the calling dialog’s parameter
/* data area (not in the parameter data area of the dialog to be opened):
1 #MYDIALOG-ID (I4)
1 #MYPARM1  (A10)
/* Pass the operands #MYPARM1 and ’MYPARM2’ to the parameters #DLG-PARM1 and
/* #DLG-PARM2 defined in the dialog to be opened:
OPEN DIALOG ’MYDIALOG’
  USING #DLG$WINDOW
  GIVING #MYDIALOG-ID
WITH #MYPARM1 ’MYPARM2’
```

When passing parameters selectively with the *PARAMETERS-clause*, a dialog may be opened as shown in the following example:

Example:

```
/* The following parameters are defined in the calling dialog's parameter
/* data area (not in the parameter data area of the dialog to be opened):
1 #MYDIALOG-ID (I4)
1 #MYPARM1 (A10)
/* Pass the operands #MYPARM1 and 'MYPARM2' to the parameters #DLG-PARM1 and
/* #DLG-PARM2 defined in the dialog to be opened:
OPEN DIALOG 'MYDIALOG'
  USING #DLG$WINDOW
  GIVING #MYDIALOG-ID
WITH PARAMETERS
   #DLG-PARM1=#MYPARM1
   #DLG-PARM2='MYPARM2'
END-PARAMETERS
```

Permanence In Creating, Passing And Checking Data

The term "permanence" is used in Natural to denote data defined in a base dialog's local data area whose existence is guaranteed throughout the whole lifetime of the dialog. Data defined in the global data area are not kept permanent because the global data area can be exchanged while the application is executed.

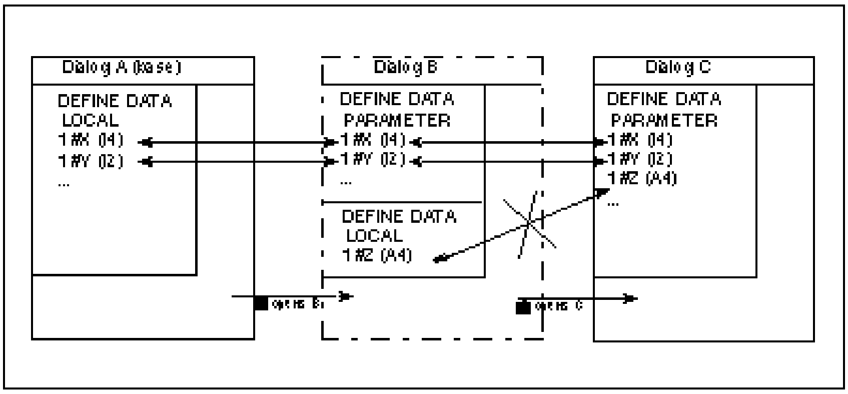
The reference to the permanent data is kept by saving the parameter data area internally during opening of the dialog. This reference is reused when

- a dialog element receives an event;
- all parameters passed from one dialog to another are permanent, provided they reference the base dialog's local data area.

Parameters are accessible

- during the before-open and after-open event processing on opening of a dialog or
- if *all of them* reference the base dialog's local data area.

The following example illustrates a case in which two parameters are kept permanently and one other is not. Assume the base dialog is dialog A. This base dialog now opens dialog B, passing parameters #X and #Y. After that, dialog B passes parameters #X and #Y on to dialog C. The #X and #Y parameters which are now in dialog C will be permanent, even if dialog B is closed. If, however, dialog B passes its own parameter #Z when opening dialog C, the parameter #Z is not permanent, because if dialog B is closed, the reference to its local data area is no longer valid. No parameter in dialog C is accessible (#Z does not reference the base dialog's local data area).



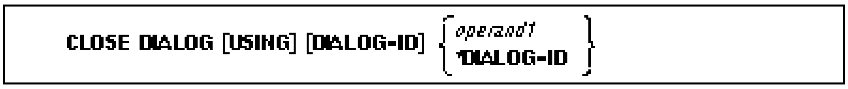
Processing Steps When Opening a Dialog

This section describes what happens when a dialog is opening. You can open a dialog either by executing it, for example from the command line, or by invoking it with an OPEN DIALOG statement.

- The dialog object is loaded and starts executing.
- The BEFORE-ANY event-handler section is executed, the value of the system variable *EVENT being OPEN.
- The BEFORE-OPEN event-handler section is executed.
- The dialog window is created as specified in the dialog editor.
- The BEFORE-ANY event-handler section is executed. *EVENT = AFTER-OPEN.
- All dialog elements are created as specified in the dialog editor.
- The dialog window and all dialogs are made visible except those that are VISIBLE = FALSE.
- The AFTER-OPEN event-handler section is executed.
- The AFTER-ANY event-handler section is executed. *EVENT = AFTER-OPEN.
- The AFTER-ANY event-handler section is executed. *EVENT = OPEN (not if the dialog’s STYLE attribute value is "Dialog Box").
-

Closing Dialogs

To close a dialog dynamically, you specify the following:



```
CLOSE DIALOG [USING] [DIALOG-ID] { operand1  }
                                 { *DIALOG-ID }
```

Operand1 is the identifier of the dialog as returned in the OPEN DIALOG statement.

Example:

```
CLOSE DIALOG *DIALOG-ID /* Close the current Dialog
```

The dialog will then be erased from the screen and removed from memory. All local data associated with the dialog will be gone.

Note: If a modal dialog is a child in a hierarchy of dialogs, the modal dialog should not close its parent(s) because this will result in a deadlock.

operand1

Operand1 is the name of the dialog to be closed.

To close the current dialog, you specify *DIALOG-ID.

Initializing Attribute Values

You can specify conditions for the opening and closing of a dialog: this applies to the before-open, after-open, and close events. These conditions can be used to initialize the attribute values in the dialog.

The following is an example of after-open event-handler code: Red foreground color is assigned to push buttons that the user must press after entering data in the associated input fields.

Example:

```
DEFINE DATA LOCAL
  ...
  1 #OK-BUTTON HANDLE OF PUSHBUTTON
  1 #CALC-BUTTON HANDLE OF PUSHBUTTON
  1 #SAVE-BUTTON HANDLE OF PUSHBUTTON
  1 #CONVERT-BUTTON HANDLE OF PUSHBUTTON
  ...
END-DEFINE
...
#OK-BUTTON.FOREGROUND-COLOUR-NAME := RED
#CALC-BUTTON.FOREGROUND-COLOUR-NAME := RED
#SAVE-BUTTON.FOREGROUND-COLOUR-NAME := RED
#CONVERT-BUTTON.FOREGROUND-COLOUR-NAME := RED
```

If you want to modify attribute values of dialog elements and of the dialog before the dialog is opened (and displayed to the end user), do not specify this in the "before open" event-handler code, because the dialog elements and the dialog window are not yet created. Instead, create the dialog with the dialog editor and set the attribute VISIBLE to FALSE in the "Dialog Attributes" window. Then modify all the attribute values in the after-open event-handler code (when the handles are available). Then make the dialog visible with VISIBLE = TRUE.

Example:

```
DEFINE DATA LOCAL
  ...
  1 #DIA-1 HANDLE OF DIALOG
  1 #OK-BUTTON HANDLE OF PUSHBUTTON
  1 #CALC-BUTTON HANDLE OF PUSHBUTTON
  ...
END-DEFINE
...
/* AFTER OPEN event-handler code section
...
#OK-BUTTON.FOREGROUND-COLOUR-NAME := RED
#CALC-BUTTON.FOREGROUND-COLOUR-NAME := RED
#DIA-1.VISIBLE := TRUE
```

How To Edit a Dialog’s Enhanced Source Code

What Is The Enhanced Source Code Format ?

The enhanced source code format enables you to edit source code that has been generated by the dialog editor. You edit enhanced source code in a program editor window. When you edit a dialog, the dialog editor stores the results in internal structures. From these structures, source code is generated when you save, stow, list or execute any other system command on the dialog. Code is also generated when you refresh the program editor’s source code window.

You can edit enhanced source code as you do any other Natural user code. The source code syntax is subject to a number of formal conventions, however. For a documentation of the enhanced source code syntax, see The Enhanced Source Code Format in the Dialog Components Manual.

When you execute a system command on a dialog you have just edited in the program editor source code window, the dialog editor updates its internal structures and refreshes the source code window.

Note: The dialog editor preserves code layout only in the user code sections, such as event handlers.

The dialog editor supports the following source formats:

- 213. This is the format generated by Natural Version 2.1.3 (New Dimension). It is supported for input only. You cannot generate 2.1.3 format with Natural Version 3.1 and Version 3.2.
- 22C. This is the format generated by Natural Version 2.2.2. In Natural for Windows and Unix/OpenVMS Version 4.1, dialogs can no longer be generated in this format. It, too, is supported for input only.
- 22D. This is the "enhanced" source-code format that from now on is the standard. It is generated for compiling, storing, and editing dialogs in Natural Version 2.2.3 and above.

The characteristics of the enhanced source code format are:

- Dialog sources are readable and printable without requiring conversion.
- Dialog sources consist only of legal and fully documented Natural syntax.
- Dialog sources can be edited textually using program editor functions such as scanning for and replacing text.
- Dialog sources can be displayed in the Natural Debugger.
- Dialog sources are larger than 213 or 22C format sources (by a factor between 1.25 and 3.5).
- Any code that can be generated with the dialog editor can also be coded manually. For example, if you "draw" a push-button control onto the user interface, the corresponding code is generated implicitly. You can also create this push-button control explicitly with the help of a source-code window that provides you with the functions of the program editor.
- You can switch between the dialog editor and the program editor by selecting the source code window or the dialog window. If you edit in either window, you need to synchronize your updates: (graphically) modifying the dialog locks the source code window and you may not make changes there. Correspondingly, if you change the source code, you may not make changes in the dialog window, which is locked. If your editor is locked, its status bar displays "Locked".

For dialogs in the old formats, this means:

- They remain unchanged until they are processed in the dialog editor. They can be compiled and executed in their old format.
- When you load them into the dialog editor, the dialogs are saved in the new format. If they are saved in the enhanced format, you must include the local data area NGULKEY1. Note that the storage size increases when the dialogs are saved.
- When you list or print them and you enable the "enhanced list mode" option, the dialogs are displayed using the enhanced source code format.

Avoiding Incompatibilities Between Dialog Editor And Program Editor

When you edit the enhanced source code format, note that some of the syntax elements accepted by the program editor are not accepted by the dialog editor. Enhanced source code editing is not intended as a new programming technique in addition to using the dialog editor:

- It may be syntactically acceptable to replace a dialog element’s numeric coordinate (a RECTANGLE-X attribute value) with a variable reference. The dialog editor, however, will not accept this when the changes are synchronized, and will prompt you when you issue a command requiring the source code.
- The dialog editor may accept a reference to a variable’s STRING attribute even if the variable is not declared, but the compiler will not accept this.

In the sections that are not user code, you should avoid such incompatibilities by adding only code that is acceptable to both the compiler and the dialog editor.

In the user code sections, such as in event-handler sections and in external or internal subroutines, your choice of programming techniques is not restricted by the dialog editor. In these sections, however, you have no visual editing support.

As a general rule, a mixed approach is often the best, especially when you use dialog-editor- generated code as a starting point.

Note: In the dialog editor, you can copy dialog elements to the clipboard and when you paste them into user code, they appear as text.

How To Use The Enhanced Source Code Format

To edit a dialog in the enhanced source code format

1. Load the dialog into the dialog editor.
2. From the "Dialog" menu, choose "Source Code".
   Or choose the "Source Code" toolbar button.
   Or press CTRL+ALT+C.

The dialog’s source code window appears and the program editor is loaded. This editor enables you to scan for text strings, replace them, and so on. For more information on how to use the program editor, see The Program Editor.

The enhanced source code format’s syntactical conventions are documented in the chapter The Enhanced Source Code Format in the Dialog Components Manual.

Enhanced source code can be listed and printed as usual. You can also scan for strings by using the Find option of the Edit menu.

Note: If you are replacing strings with this option, this can make a dialog source incompatible with the dialog editor.

How Dialogs, Controls and Items Are Related Hierarchically

Dialogs and their dialog elements are organized hierarchically. Typically, the dialog window contains a number of controls. The controls are children of the window or of other controls which are capable of acting as containers. A control may contain a number of items. For example, a list-box control may contain several list-box items. The control is the parent of the items.

The dialogs themselves are also organized hierarchically. Every time the OPEN DIALOG statement is specified, the parent of the newly created dialog must be provided as a parameter. This parameter may be NULL-HANDLE or the handle of an existing dialog. If NULL-HANDLE is provided, the dialog belongs to the desktop rather than to any other dialog. This means that the dialog can be closed and minimized independently of any other dialog in the application. A dialog having an existing dialog as parent is closed or minimized when the parent dialog is closed or minimized.

The first dialog in an application plays a special role and is sometimes called the base dialog. When the base dialog is closed, all other dialogs in the application are also closed, whether they are children of the base dialog or not.

All children on one hierarchical level are sorted in the sequence of their creation. Each dialog element therefore always "knows" its parent, its predecessor and successor (on the same hierarchical level), and its first and last child (if present). You can retrieve this information by using the following attributes:

- PARENT
- PREDECESSOR
- SUCCESSOR
- FIRST-CHILD
- LAST-CHILD

These attributes contain handle values of dialog elements. If their value is NULL, the dialog element has no parent, successor, or child. The following example demonstrates how to go through all dialog elements of a dialog.

Example 1:

```
1 #CONTROL HANDLE OF GUI

#CONTROL := #DLG$WINDOW.FIRST-CHILD
REPEAT UNTIL #CONTROL = NULL-HANDLE
  ...
  #CONTROL := #CONTROL.SUCCESSOR
END-REPEAT
```

List-box controls and list-box items contain an additional attribute:

SELECTED-SUCCESSOR can be set for either the list-box control itself or for any of its items. It points to the next selected item in a list-box control. For the list-box control itself, it points to the first selected item.

Example 2:

```
1 #ITEM HANDLE OF LISTBOXITEM

#ITEM := #LISTBOX.SELECTED-SUCCESSOR
REPEAT UNTIL #ITEM = NULL-HANDLE
  ...
  #ITEM := #ITEM.SELECTED-SUCCESSOR
END-REPEAT
```

The above example is the query necessary to find all selected items in a list-box control where multiple selection is allowed (MULTI-SELECTION attribute).

How To Define Dialog Elements

Dialog elements are uniquely identified by a handle. A handle is a binary value that is returned when a dialog element is created. A handle must be defined in a DEFINE DATA statement of the dialog.

You can define a handle

- by creating a dialog or a dialog element with the dialog editor; in this case, the handle definition is generated;
- by explicitly entering the definition in a global, local, or parameter data area of the dialog;
- by explicitly entering the definition in a subprogram or a subroutine.

Note: Handles of ActiveX controls are defined in a slightly different way than the standard handle definition described below. This is described in Working with ActiveX Controls.

A handle is defined inside a DEFINE DATA statement in the following way:

```
level handle-name [(array-definition)] HANDLE OF dialog-element-type
```

Handles may be defined on any *level*.

Handle-name is the name to be assigned to the handle; the naming conventions for user-defined variables apply.

Dialog-element-type is the type of dialog element. Its possible values are the values of the TYPE attribute. It may not be redefined and not be contained in a redefinition of a group.

Examples:

```
1 #SAVEAS-MENUITEM HANDLE OF MENUITEM
1 #OK-BUTTON (1:10) HANDLE OF PUSHBUTTON
```

When you have defined a handle, you can use the *handle-name* with handle attribute operands in those Natural statements where an operand may be specified. With handle attribute operands, you can, for example, dynamically query, set, or modify attribute values for the defined *dialog-element-type*. This is the most important programming technique in the dialog editor. For details, see the section How To Manipulate Dialog Elements.

If there is a dialog element handle of the same name in two different dialogs, the PARENT attribute ensures that Natural knows the difference between the two handles (two different PARENT values). Handles may be passed as parameters or may be assigned from one handle variable to another.

HANDLE OF GUI

In addition to the handle types referring to one dialog element, the generic handle type HANDLE OF GUI is available. In event-handler code, you can use HANDLE OF GUI to refer to the handle of any type of dialog element.

This can be useful, for example, if you are querying an attribute value in all dialog elements on one level: you go through the dialog elements one after the other; in the course of this query, it is not clear which type of dialog element is going to be queried next. Then a GUI handle makes it possible to query the next dialog element regardless of its type. This saves a lot of coding, because otherwise, you would have to query the attribute's value of each dialog element separately.

Example:

```
...
1 #CONTROL HANDLE OF GUI
...
#CONTROL := #DLG$WINDOW.FIRST-CHILD
REPEAT UNTIL #CONTROL = NULL-HANDLE
  ...
  #CONTROL := #CONTROL.SUCCESSOR
END-REPEAT
```

NULL-HANDLE

The HANDLE constant "NULL-HANDLE" may be used to query, set or modify a NULL value of a HANDLE. Such a NULL value means that the dialog element is nonexistent (even if it has been created explicitly).

Example:

```
DEFINE DATA PARAMETER
  1 #PUSH HANDLE OF PUSHBUTTON
END-DEFINE
...
IF #PUSH = NULL-HANDLE
...
```

The HANDLE constant "NULL-HANDLE" represents the NULL value of a HANDLE variable or of an attribute with format HANDLE. For handle variables, the value indicates that the expression *handle.attribute* refers to the global attribute list. For attributes, this value indicates that no value is currently set.

How To Manipulate Dialog Elements

To manipulate dialog elements, Natural provides you with handle attribute operands. You use handle attribute operands wherever an operand may be specified in a Natural statement. This is the most important programming technique in event-handler code.

Important: You must have defined a handle.

Note: ActiveX controls are manipulated in a slightly different way than the standard way described below. This is described in Working with ActiveX Controls.

Handle attribute operands may be specified as follows:

```
handle.name - attribute.name [(index-specification)]
```

The *handle-name* is the handle of the *dialog-element-type* as defined in the HANDLE definition of the DEFINE DATA statement.

The *attribute-name* is the name of an attribute which has to be valid for the *dialog-element-type* of the handle.

Examples:

```
1 #PB-1 HANDLE OF PUSHBUTTON   /* #PB-1 is a handle-name of the
                               /* dialog-element-type PUSHBUTTON
RESET #PB-1.STRING...          /* #PB-1.STRING is the handle attribute operand
                               /* where STRING is a valid attribute-name of the
                               /* dialog-element-type PUSHBUTTON

1 #RB-1(1:5) HANDLE OF RADIOBUTTON /* #RB-1 is an array of five RADIOBUTTONs
IF #RB-1.CHECKED(3) = CHECKED      /* If the third radio-button control is
   THEN...                         /* checked ...
```

Querying, Setting and Modifying Attribute Values

In most applications, it will be necessary

- to set an attribute value before creating the dialog element,
- to modify the value after creating the dialog element, and
- to query an attribute value.

In some cases, it may be necessary to modify and query some attributes during processing, for example to query the checked/not checked state of a radio-button control or to disable (= modify) a menu item.

You can do that, for example, in the ASSIGN, MOVE or CALLNAT statements.

Examples:

```
1 #PB-1 HANDLE OF PUSHBUTTON      /* #PB-1 is a handle-name of the
...                               /* dialog-element-type PUSHBUTTON
#PB-1.STRING:= 'MY BUTTON'        /* Set or modify the value of the STRING
                                  /* attribute to 'MY BUTTON'
#TEXT:= #PB-1.STRING              /* Query the value of the STRING attribute
                                  /* and assign the value to #TEXT
CALLNAT 'SUBPGM1' #PB-1.STRING    /* Query the value of the STRING attribute
                                  /* and pass it on to the subprogram
```

When you use the *handle-name* variable only on the left side of the statements, as in the first of the three examples above, the attribute value is set or modified, that is, it is assigned the value of the specified *operand*.

When you use the *handle-name* variable on the right side of the statements, as in the second example, the attribute value is queried, that is, the value is assigned to the *operand*.

Once a handle has been defined (either explicitly in specified Natural code, or implicitly with the dialog editor), it can be used with most Natural statements. However, only a specific set of attributes can be queried, set or modified for a particular dialog element. To find out which values an attribute can have, see the chapter Attributes in the Dialog Components Manual.

Although an exact data type is specified for the values of most attributes, it is sufficient to supply move-compatible values to a handle attribute operand. The rules are the same as those for Natural variables.

Restrictions

Handle attribute operands must not be used in the following statements:

AT BREAK, FIND, HISTOGRAM, INPUT, READ, READ WORK FILE.

User-defined variables can be used instead.

Numeric/Alphanumeric Assignment

If you assign numeric operands to alphanumeric attributes, the values of these attributes will be in a non-displayable format. The Natural arithmetic assignment rules apply.

If you need a displayable format, you can use MOVE EDITED.

Examples:

```
#PB-1.STRING:= -12.34                        /* Non-displayable format
MOVE EDITED #I4 (EM = -Z(9)9) TO #PB-1.STRING /* Displayable format
```

The following edit masks may be used for the various format/length definitions of numeric operands:

Format/Length	Edit Mask
I1	-ZZ9
I2	-Z(5)9
I4	-Z(9)9
N*n.m*/P*n.m*	-Z(*n*).9(*m*)

How To Create and Delete Dialog Elements Dynamically

Dialog elements are usually added to a dialog by means of the dialog editor. However, they can also be created and deleted dynamically. This may be done, for example, when the layout of a dialog is strongly context-sensitive.

A dialog element is created dynamically with the ADD action of the PROCESS GUI statement. This action returns a handle to the newly created dialog element. As soon as the dialog element is created, this handle points to a set of attributes specified for the dialog element just created.

Note: ActiveX controls are created in a slightly different way than the standard way described below. This is described in Working with ActiveX Controls.

For more information on the actions available, and on the parameters that can be passed, see the chapter Executing Standardized Procedures.

Global Attribute List

By modifying any handle attribute operand of the form "*handlename.attributename*" (for example, #PB-1.STRING), you change an attribute value of the specific dialog element. As long as the dialog element is not yet created and the handle variable has its initial value (NULL-HANDLE), the handle attribute operand "*handlename.attributename*" refers to the global attribute list.

The global attribute list is a collection of all attributes defined for any dialog element. Natural contains one such collection. Whenever a dialog element is created, it "inherits" its attributes from this global attribute list. It does not inherit them when you create the dialog element with the PROCESS GUI statement action ADD using the WITH PARAMETERS option.

Creating Dialog Elements Statically and Dynamically

To define a dialog element statically (in the dialog editor), with an individual set of attributes, you must first set the attributes in the global attribute list to the desired values and then create the dialog element. After creation, the values of the attributes in the global attribute list remain intact. The next created dialog element gets the same attributes from the global attribute list as the previous one, except those that have been modified.

The status of the global attribute list as found in the "after open" event handler is influenced by the dialog elements defined statically. Therefore, before you start creating dialog elements dynamically in the "after open" event handler, you should reset the attributes by means of the PROCESS GUI action RESET-ATTRIBUTES to prevent your dialog elements from inheriting unexpected values from the global attribute list. If you want to avoid this inheritance problem, use the PROCESS GUI statement action ADD with the WITH PARAMETERS option.

Unexpected values may also result from having attribute values that mean different things if used by different types of dialog elements. For example, the value "s" of the attribute STYLE means "scaled" for the dialog element type bitmap control but "solid" for the dialog element type line control.

The PROCESS GUI action ADD is used to define a dialog element dynamically. This clause of the PROCESS GUI statement enables you to specify the attribute values within the statement. The inheritance of attributes from the global attribute list does not affect the PROCESS GUI statement action ADD. The attributes specified in the statement are transferred to the global attribute list before the action ADD is performed.

Note: When you use the PROCESS GUI statement with Parameter Clause 2 of the ADD action, the global attribute list is not used or affected. For parameters which are needed to create the dialog element, but which were not specified in the WITH PARAMETERS section of the PROCESS GUI action ADD statement, the default value is taken. Apart from these, only the parameters which are passed explicitly in the parameter list are used to create the dialog element.

To create list-box and selection-box items dynamically, it may be more convenient to use the PROCESS GUI action ADD-ITEMS. This allows you to insert several items at a time.

Example:

```
/* #PB-A inherits the current settings of the global attribute list
#PB-A.STRING := ’TEST1’
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-A
#PB-B.STRING := ’TEST2’
/* #PB-B has the same attributes as #PB-A except STRING. This leads to #PB-B
/* covering #PB-A.
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-B
COMPUTE #PB-C.RECTANGLE-Y = #PB-B.RECTANGLE-Y + #PB-C.RECTANGLE-H + 20
/* #PB-B has the same attributes as #PB-A except RECTANGLE-Y
/* #PB-C will be located 20 pixels below #PB-B
PROCESS GUI ACTION ADD WITH #DLG$WINDOW PUSHBUTTON #PB-C
```

To delete dialog elements dynamically, you use the PROCESS GUI action DELETE. You can also use this technique to delete dialog elements created with the dialog editor (at design time). You should, however, avoid using the handle of the deleted dialog element because this is invalid.

Dialog elements often do not have to be created dynamically. In some cases, it is sufficient to make dialog elements VISIBLE = TRUE and VISIBLE = FALSE, depending on the context. This technique is more efficient and easier to handle. It also enables you to "insert" dialog elements anywhere in the navigation sequence.

Example:

```
DEFINE DATA LOCAL
   ...
   1 #PB-1 HANDLE OF PUSHBUTTON
   ...
END-DEFINE
...
#PB-1.VISIBLE := FALSE
...
IF...                        /* Logical condition
   #PB-1.VISIBLE := TRUE
END-IF
```

How to Handle Events of Dynamically Created Dialog Elements

When a dialog element is created dynamically, you cannot use the dialog editor to associate events to it. Instead, you must handle all events of all dynamically created dialog elements in the DEFAULT event. In this event, you must filter out which event occurred for which dialog element. The code for this is similar to the code generated by the dialog editor. The general structure is:

Example:

```
DECIDE ON FIRST *CONTROL
VALUE #PB-A
   DECIDE ON FIRST *EVENT
      VALUE ’CLICK’
         /* Click event-handler code
      NONE
      IGNORE
   END-DECIDE
VALUE #PB-B
   ...
VALUE #PB-C
   ...
END-DECIDE
```

In the case of event code for dynamically created ActiveX controls, *where event parameters are used*, it is necessary to precede the event code with an OPTIONS 2 statement containing the name of the event, otherwise the compiler will not be able to process parameter references (e.g., #OCX-1.<<PARAMETER->>) successfully. However, in contrast to the implicit generation of the OPTIONS statement by the Dialog Editor for events for statically created controls, no OPTIONS 3 statement should be coded in this case. Otherwise the Dialog Editor would falsely interpret the OPTIONS 3 statement as the end marker for the DEFAULT event, resulting in a scanning error on attempting to load the dialog.

Example:

```
DECIDE ON FIRST *CONTROL
VALUE #OCX-1  /* MS Calendar control
  DECIDE ON FIRST *EVENT
   VALUE ’-602’ /* DispID for KeyDown event
     OPTIONS 2 KeyDown
      /* KeyDown event-handler code containing parameter
      /* access (e.g. #OCX-1.<>)
     NONE
      IGNORE
 END-DECIDE
...
END-DECIDE
```

How To Enable and Disable Dialog Elements

During end-user interaction, it may be implicitly clear that certain dialog elements must not be used. For example, if a dialog requiring personnel data contains a group of radio-button controls for marital status and an input-field control for date of marriage, the input-field control must be disabled whenever the marital status is other than "married".

There are two ways to do this:

- Use Natural code to enable/disable a dialog element dynamically.
- Use the dialog editor (to disable a dialog element initially).

The first method is used more often.

The Natural code might look like this:

Examples:

```
/*First alternative
...
IF #RB-1.ENABLED = TRUE     /* Logical condition
   #IF-1.ENABLED := TRUE    /* Set ENABLED to TRUE
END-IF
...
/*Second alternative
#PB-1.ENABLED := #RB-1.ENABLED
```

When you use the dialog editor, you set the attribute ENABLED to TRUE by marking the "Enabled" entry in the dialog element’s attributes window.

To disable editing in input-field controls, selectionbox controls and edit area controls, it is not always necessary to disable these dialog elements entirely. It may be sufficient to make them MODIFIABLE = FALSE.

Defining and Using Context Menus

As from Natural v4.1.1, it is possible to create context menus for use within Natural applications. The context menus can be completely static (i.e., the menu contents are known in advance and can be built via the dialog editor) or wholly or partially dynamic (i.e., the menu contents and/or state depend on the runtime context and are not completely known at design time).

Construction

A context menu is very similar in concept to a submenu. Therefore, the same menu editor is used for editing a context menu as is used for editing a dialog’s menu bar. Menu items can be added to context menus, and events associated with them, in exactly the same way as for menu-bar submenus. There are no functional differences to the menu-bar editor, except that the ’OLE’ combo box (which is applicable only to top-level menu-bar submenus) will always be disabled. It should be noted, however, that any accelerators defined for context menu items will be globally available as long as that menu item exists. Furthermore, the accelerator will trigger the menu item for which it is defined even if the context menu is not being displayed or if the focus is on a control using a different context menu or no context menu at all.

The context-menu editor may be invoked via either a new menu item, ’Context menus...’ on the ’Dialog’ menu, or via its associated accelerator (CTRL+ALT+X by default), or toolbar icon. However, because the context-menu editor can only edit one context menu editor at a time, the context-menu editor is not invoked directly. Instead, the Dialog Context Menus window is shown, where operations on the context menu as a whole are made, and from which the menu editor for a given (selected) context menu can be invoked.

Internally, in order to distinguish between submenus and context menus, context menus have a new type, CONTEXTMENU. Otherwise, the generated code in both cases is identical. Here is some sample code illustrating the statements used to build up a simple context menu containing two menu items:

```
/* CREATE CONTEXT MENU ITSELF:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #CONTEXT-MENU-1
  TYPE = CONTEXTMENU
  PARENT = #DLG$WINDOW
END-PARAMETERS GIVING *ERROR
/* ADD FIRST MENU ITEM:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-1
  TYPE = MENUITEM
  DIL-TEXT = ’Invokes the first item’
  PARENT = #CONTEXT-MENU-1
  STRING = ’Item 1’
END-PARAMETERS GIVING *ERROR
/* ADD SECOND MENU ITEM:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-2
  TYPE = MENUITEM
  DIL-TEXT = ’Invokes the second item’
  PARENT = #CONTEXT-MENU-1
  STRING = ’Item 2’
END-PARAMETERS GIVING *ERROR
```

Note that if context menus or context-menu items are created dynamically in user-written code, the context menu or menu items will not be visible to the dialog editor. For example, the dynamically created menu item will not be visible in the context-menu list box, and the dynamically created menu items will not be visible in the context-menu editor.

Association

After creating a context menu, the context menu needs to be associated with a Natural object. Context menus are supported for almost all controls types capable of receiving the keyboard focus and for the dialog window itself. The full list includes ActiveX controls, bitmaps, canvasses, edit areas and input fields, list boxes, push buttons, radio buttons, scroll bars, selection boxes, table controls, toggle buttons, standard and MDI child windows, and MDI frame windows.

For all object types supporting context menus, the corresponding attribute dialogs in the dialog editor include a read-only combo box listing all context menus created by the dialog editor, together with an empty entry. The selection of the empty entry implies that no context menu is to be used for this object, and is the default.

Internally, the association is achieved by a new attribute, CONTEXT-MENU , which should be set to the handle of a context menu. This attribute can be assigned at or after object creation time, and defaults to NULL-HANDLE if not specified, indicating the absence of a context menu. For context menus created by the dialog editor, the context menu is specified at control creation time as illustrated below:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
  HANDLE-VARIABLE = #LB-1
  TYPE = LISTBOX
  RECTANGLE-X = 585
  RECTANGLE-Y = 293
  RECTANGLE-W = 142
  RECTANGLE-H = 209
  MULTI-SELECTION = TRUE
  SORTED = FALSE
  PARENT = #DLG$WINDOW
  CONTEXT-MENU = #CONTEXT-MENU-1
  SUPPRESS-FILL-EVENT = SUPPRESSED
END-PARAMETERS GIVING *ERROR
```

The same syntax can also be used for controls created in user-written event code. In other cases, where the control was created by the dialog editor but the context menu was not, the context menu attribute must be assigned to the control after its creation, e.g., in the dialog’s AFTER-OPEN event:

```
/* CONTEXT MENU SPECIFIED AFTER CREATION:

#LB-2.CONTEXT-MENU := #CONTEXT-MENU-2
```

Note that a context menu is not destroyed when an object using it is destroyed. Instead, it is destroyed when its parent object (typically, the dialog for which the context menu was defined) is destroyed. Similarly, the assignment of a new menu handle to the CONTEXT-MENU attribute where one is already assigned does not result in the previous context menu being destroyed. Thus, using the above examples, neither of the following statements results in CONTEXT-MENU-1 being destroyed:

```
PROCESS GUI ACTION DELETE WITH #LB-1             /* #CONTEXT-MENU-1 LIVES ON

#LB-1.CONTEXT-MENU := #CONTEXT-MENU-2            /* SAME HERE
```

Invocation

The invocation of static context menus is transparent to the application. The tracking of the context menu and the triggering of the events associated with the menu items is done by Windows and Natural. The context menu is always displayed at the current mouse cursor position. Therefore, there are no new PROCESS GUI statements for displaying context menus.

However, in order to support dynamic context menus or static context menus that need to be modified at runtime (e.g. to disable or check particular menu items before the context menu is displayed), context menus and submenus receive a BEFORE-OPEN event. This applies to submenus belonging to a menu bar as well as to submenus belonging to a context menu. In addition, it is possible to suppress this event via the use of a new attribute, SUPPRESS-BEFORE-OPEN-EVENT, which defaults to SUPPRESSED. Assuming the event is not suppressed, the BEFORE-OPEN event will be triggered immediately before a context menu is displayed. This gives the application the chance to modify the context menu according to the current program state. For example, menu items could be added or deleted, or particular menu items grayed or checked. Here is some sample code for the BEFORE-OPEN event:

```
/* DELETE FIRST MENU ITEM:
PROCESS GUI ACTION DELETE WITH #MITEM-1
/* CHECK SECOND MENU ITEM:
#MITEM-2.CHECKED := CHECKED
/* DISABLE THIRD MENU ITEM:
#MITEM-3.ENABLED := FALSE
/* INSERT NEW MENU ITEM BEFORE #MITEM-3:
PROCESS GUI ACTION ADD WITH PARAMETERS
  HANDLE-VARIABLE = #MITEM-4
  TYPE = MENUITEM
  DIL-TEXT = 'Invokes the first item'
  PARENT = #CONTEXT-MENU-1
  STRING = 'Item 3'
  SUCCESSOR = #MITEM-3
END-PARAMETERS GIVING *ERROR
```

For context menus not created by the dialog editor, the handling of the BEFORE-OPEN event must be done in the DEFAULT event for the dialog. Note also that if a control or dialog is disabled, no context menu is displayed, and the BEFORE-OPEN event is also not triggered. The same applies if the context menu itself is disabled. For example:

```
#CONTEXT-MENU-1.ENABLED := FALSE        /* DISABLE CONTEXT MENU DISPLAY
```

Note that it is possible to disable the context menu in this way during the BEFORE-OPEN event, allowing selective disabling of the context menu depending on the mouse cursor position within the control. For example, it might be desired to only display a context menu if the mouse cursor is over a selected list-box item. Determining whether this is the case is possible via the use of two PROCESS GUI ACTION calls:

- INQ-CLICKPOSITION has been extended to controls other than bitmaps and canvasses to return the (X, Y) position of the right mouse button click within the control. This is updated immediately prior to the sending of the BEFORE-OPEN event.
- INQ-ITEM-BY-POSITION. This allows translation of the relative co-ordinate returned by INQ-CLICKPOSITION applied to a list box to the corresponding item.

As an example of the use of these two new actions, consider the situation where we want to detect whether the cursor was over a selected list-box item when the right mouse button was pressed in order to determine whether to display a context menu or not. This can be achieved by the following code in the BEFORE-OPEN event of the associated context menu:

```
PROCESS GUI ACTION INQ-CLICKPOSITION WITH
   #LB-1 #X-OFFSET #Y-OFFSET
PROCESS GUI ACTION INQ-ITEM-BY-POSITION WITH
   #LB-1 #X-OFFSET #Y-OFFSET #LBITEM
#MENU = *CONTROL
IF #LBITEM = NULL-HANDLE                /* NO ITEM UNDER (MOUSE) CURSOR */
  #MENU.ENABLED := FALSE
ELSE
  IF #LBITEM.SELECTED = FALSE           /* ITEM UNDER CURSOR DESELECTED */
     #MENU.ENABLED := FALSE
  ELSE                                  /* ITEM UNDER CURSOR IS SELECTED */
     #MENU.ENABLED := TRUE
  END-IF
END-IF
```

In some cases, it may be desired to automatically select the item under the mouse cursor if it is not already selected, clearing any existing selection. For list boxes, it is possible to achieve this by using the new AUTOSELECT attribute, either directly or via the new 'Autoselect' check box (see previous bitmap) in the List Box Attributes window in the dialog editor. If this attribute is set to TRUE, Natural will automatically update the selection before sending the BEFORE-OPEN event, if the context menu was invoked over an unselected list-box item.

For table controls, any change in the selection must be done via the application itself in the BEFORE-OPEN event. To make this possible, another new PROCESS GUI ACTION has been introduced for table controls:

- TABLE-INQUIRE-CELL. This returns the cell’s row and column number (starting from 1) for a relative (X, Y) position within the table. This position can (and would typically be) the position returned by a previous call to PROCESS GUI ACTION INQ-CLICKPOSITION.

Sharing of Context Menus

It is of course possible to associate the same context menu with more than one object (i.e., control or dialog). For example:

```
#LB-1.CONTEXT-MENU := #CTXMENU-1
#LB-2.CONTEXT-MENU := #CTXMENU-1
```

In such a scenario, we need to be able to determine for which control the context menu was invoked. We cannot use *CONTROL in the BEFORE-OPEN event, because this will contain the handle of the context menu itself. Instead, it is necessary to inquire which control has the focus, since Natural automatically places the focus on the control for which the context menu is being invoked. Here is some sample BEFORE-OPEN event code illustrating the use of this technique:

```
PROCESS GUI ACTION GET-FOCUS WITH #CONTROL
DECIDE ON FIRST VALUE OF #CONTROL
  VALUE #LB-1
     #MITEM-17.ENABLED := FALSE
  VALUE #LB-2
     #MITEM-17.CHECKED := CHECKED
  NONE
     IGNORE
END-DECIDE
```

System Variables

Whenever you specify an event to occur with a given dialog element, the dialog editor generates code containing the Natural system variables *CONTROL, *DIALOG-ID and *EVENT.

During the processing, *CONTROL contains the dialog element’s handle, *EVENT contains the event name and *DIALOG-ID identifies an instance of a dialog.

You can reference these system variables whenever you enter Natural code within the dialog editor. If, for example, the end user clicks on a push-button control and the event handler calls a shared subroutine, you can use these system variables as logical condition criteria to trigger the subroutine.

For further details on these system variables, see the Natural Reference Manual.

Generated Variables

#DLG$PARENT

You use this generated variable of type "user" to work with MDI child windows, for example. When you create a dialog, Natural generates this variable in order to hold the handle of the parent dialog. In event-handler code, you can, for example, use this variable to open an MDI child dialog from another MDI child dialog, as shown below.

Note: You should not use names for user-defined variables that begin with #DLG$ to avoid conflicts with generated variables.

Example:

```
OPEN DIALOG 'MDICHILD' #DLG$PARENT #CHILD-ID
```

#DLG$WINDOW

You use this generated variable to dynamically set the attributes within a dialog. When you create a dialog, Natural generates this variable in order to hold the handle of the dialog window. #DLG$WINDOW is the default name of this variable; you may change it by overwriting the "Name" entry in the upper left of the dialog's attributes window. In event-handler code, you can, for example, use this variable to minimize the dialog window if certain logical condition criteria are met, as shown below.

#DLG$WINDOW represents the graphical user interface aspects of a dialog, while the *DIALOG-ID system variable represents the runtime aspects. *DIALOG-ID must be used in OPEN DIALOG, CLOSE DIALOG and SEND EVENT statements.

Note: You should not use names for user-defined variables that begin with #DLG$ to avoid conflicts with generated variables.

Example:

```
...
IF ...
   #DLG$WINDOW.MINIMIZED := TRUE
END-IF
...
```

Message Files and Variables as Sources of Attribute Values

Most dialog elements have a STRING attribute. As an alternative to specifying the attribute value by typing in the text in the "String" entry of the attributes window, you can select a variable or a message file number from which the text is taken at runtime. In this case, the attribute value is determined by the variable’s current value or the selected message file at the dialog element’s creation time. You can also specify attribute sources for the BITMAP-FILE-NAME, DIL-TEXT and ACCELERATOR attributes.

To select a message file number or specify a variable

1. Invoke the dialog element’s attribute window.
2. Push the "Source" button to the right of the "String" entry.

The "Attribute Source" dialog box appears. The default attribute source is "Constant"; you can also enter the number of the message file, or enter the variable name.

Note: If you are using an integer variable as the source of an attribute value, note that at runtime, the message with the corresponding number from your message file will be displayed. To avoid this, you can MOVE the contents of this integer variable to a variable of format N, for example.

Triggering User-Defined Events

Aside from standard events, such as before-open, you may define user-defined events for dialogs. User-defined events are useful whenever it is necessary for one dialog to cause an action to occur in another dialog.

A user-defined event occurs whenever you have specified a SEND EVENT statement in dialog A with the name of a user-defined event in the target dialog B. This target dialog B for which you wish to trigger the user-defined event must already be active. You can activate dialog B by using the OPEN DIALOG statement. If you do not issue the OPEN DIALOG statement first, the SEND EVENT statement will cause a runtime error.

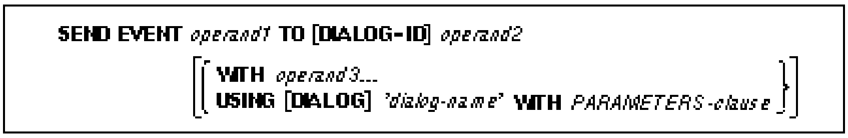
You can define your own events for dialogs by pressing the "New" button in the "Events" dialog event handler menu or from the dialog’s context menu. Enter any name for your newly-defined event and specify the corresponding event section. It is recommended that this name begin with "#" to distinguish your event from predefined events.

During execution of an event handler, the SEND EVENT statement triggers the user-defined event handler in a different dialog. After this user-defined event handler has been executed, control will be returned to the previous dialog, whose execution will resume at the statement following the SEND EVENT statement. This can be compared to a CALLNAT statement that causes a subprogram to be executed.

Similar to the OPEN DIALOG statement, parameters may be passed to the dialog. In order to pass parameters selectively (*PARAMETERS-clause*), you have to specify the name of the dialog in addition to the identifier of the dialog *(operand2)*.

The SEND EVENT statement must not trigger an event in a dialog that is about to process an event. This is the case, for example, when dialog A sends an event to dialog B and the event handler in dialog B sends an event to dialog A which has not yet finished its event handling. A similar case is when dialog A opens dialog B and the before-open or after-open event contains a SEND EVENT back to dialog A.

To trigger a user-defined event, you specify the following syntax:



```
SEND EVENT operand1 TO [DIALOG-ID] operand2
        [ WITH operand3...                                          ]
        [ USING [DIALOG] 'dialog-name' WITH PARAMETERS-clause        ]
```

Operands

Operand1 is the name of the event to be sent.

Operand2 is the identifier of the dialog receiving the user-defined event and must be defined with format/length I4. You can retrieve this identifier, for example, by querying the value of #DLG$PARENT.CLIENT-DATA.

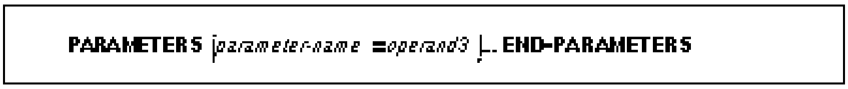
Passing Parameters to the Dialog

It is possible to pass parameters to the dialog receiving the user event.

As *operand3* you specify the parameters which are passed to the dialog.

With the *PARAMETERS-clause*, parameters may be passed selectively.

PARAMETERS-clause



```
PARAMETERS [parameter-name =operand3 ]... END-PARAMETERS
```

Note: You may only use the PARAMETERS-clause if the target dialog is cataloged.

Dialog-name is the name of the dialog receiving the user-defined event.

When you use only *operand3* to pass parameters, it might look like this:

Example:

```
/* The following parameters are defined in the dialog’s
/* parameter data area:
1 #DLG-PARM1 (A10)
1 #DLG-PARM2 (A10)
1 #DLG-PARM3 (A10)
1 #DLG-PARM4 (A10)
/* When sending the user-defined event, pass the operands #MYPARM1 ’MYPARM2’ to
the parameters #DLG-PARM1 and #DLG-PARM2:
SEND EVENT ’MYEVENT’ TO #DLG$DIA-ID WITH #MYPARM1 ’MYPARM2’
```

When you use the *PARAMETERS-clause*, the user-defined event might look like this:

Example:

```
/* The following parameters are defined in the dialog’s
/* parameter data area:
1 #DLG-PARM1 (A10)
1 #DLG-PARM2 (A10)
1 #DLG-PARM3 (A10)
1 #DLG-PARM4 (A10)
/* When sending the user-defined event, the operand #MYPARM2 is passed to the
/* parameter #DLG-PARM2 and the operand ’MYPARM3’ is passed to the parameter
/* #DLG-PARM3:
SEND EVENT ’MYEVENT’ TO #DLG$DIA-ID
  USING DIALOG ’MYDIALOG’
  WITH PARAMETERS
    #DLG-PARM3=’MYPARM3’
    #DLG-PARM2=#MYPARM2
  END-PARAMETERS
```

To avoid format/length conflicts between operands passed and their parameter definitions, see the BY VALUE option of the DEFINE DATA statement in the Natural Statements Manual.

Suppressing Events

If an event occurs, normally an event handler will be triggered. It may, however, sometimes be necessary to dynamically suppress the execution of the event-handler code whenever the event has occurred. For example, if you want to modify the string of an input field control within the change-event handler, you must suppress the change event before modification to avoid an infinite loop because the modification itself triggers a change event.

The event-handler code may look like this:

Example:

```
...
IF...                                            /* Logical condition criteria
   #IF-1.SUPPRESS-CHANGE-EVENT := SUPPRESSED     /* Suppress the event
END-IF
...
```

By default, the dialog editor generates code to suppress all events for which no event handler code has been entered. In the dialog editor, you can also suppress an event with the Suppress option in the "Events..." dialog box.

If you suppress an event, the before-any and after-any events are also suppressed for this event.

Menu Structures, Toolbars and the MDI

Creating a Menu Structure

A menu structure consists of three types of dialog elements:

- menu-bar controls,
- menu items,
- submenu controls.

A menu structure has one menu-bar control consisting of several menu items. The menu bar with its items is displayed directly beneath the window’s title bar. Each menu item may be simple or may represent a submenu control, which allows you to pull down several menu items grouped vertically. Therefore, submenu controls may contain items representing a submenu control one level lower. A submenu control becomes visible when the representing item in the menu-bar control or the parent submenu control is clicked upon.

There are two ways to create menu structures:

- Use the dialog editor; or
- use Natural code.

If you use the dialog editor

1. Check the "Menu Bar" entry in the dialog’s attribute window. Click OK.
   When you go back to the dialog, a dummy menu-bar control appears.
2. Double-click on the dummy menu-bar control, or from the Natural Menu, select "Dialog > Menu Bar", or use CTRL+M.
   The "Dialog Menu Bar" dialog box appears. This dialog box is divided into three group frames: menu bar, selected submenu and selected menu item.
3. In the selected menu items group frame, use "New" to add a menu item behind the selected position, or at the beginning. Now use the selected menu-item group frame to modify attribute values or add event handlers to the new menu item.

Normal menu items have a click event whose code is executed when the end user clicks on the menu item.

Note: The MENU-ITEM-TYPE of the menu item can also be "Separator", in which case the item is no text item.

If you use Natural code

1. Create a Menu Bar with the PARENT attribute set to ’NULL-HANDLE’ or ’*windowhandle*’.
2. To create a simple menu item: the PARENT attribute must have the value ’*menubarhandlename*’.
3. To create a submenu control: the submenu control’s PARENT attribute must have the value ’NULL-HANDLE’ or ’*windowhandlename*’. Then create a menu item with PARENT = ’*menubarhandlename*’ and MENU-HANDLE = *’submenuhandlename’*.
4. Then associate the menu bar with a dialog window by updating the window’s MENU-HANDLE attribute with the handle of the menu bar as set in the first step.
5. The event handling for the dynamically created menu items must be done in the default event handler, as described in the section How to Create and Delete Dialog Elements Dynamically.

The PARENT attribute determines when the menu bar or the submenu control will be destroyed. When PARENT = ’*windowhandlename*’, the menu bar/the submenu control will be destroyed when the window is destroyed. This is the default setting, which is also used by the dialog editor. If PARENT = NULL-HANDLE, the menu bar/the submenu control will be destroyed only when the application is terminated.

If you define the menu structure's handles inside a global data area, you can share these definitions among several dialogs.

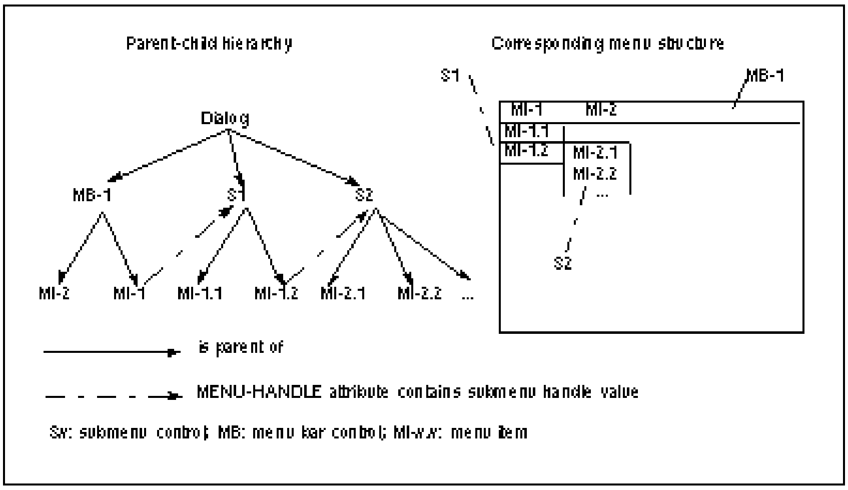
To build the above menu structure

1. Define the handles of the menu-bar control, the menu items, and the submenu control(s) as the user-defined variables in the handler of the applicable event.
2. Create the controls and items by assigning values to the attributes (PARENT, ...) and by executing the PROCESS GUI statement action ADD.
3. Create the controls and items in the sequence menu-bar control, submenu control with menu items.
4. Insert the controls and items in the sequence submenu control into menu-bar control, and menu-bar control into dialog window.

You can study how to build a menu structure in code by using the enhanced dialog list mode to list a dialog with an editor-built menu. To get a code model for creating a menu item, create a menu-bar control with the dialog editor, go to the menu-bar control attributes window, cut a menu item and paste it into any chosen event-handler section. The generated code for the menu item appears.

Parent-Child Hierarchy in Menu Structures

Sometimes, it is necessary to use code for going through each element in a menu structure. For menus, the parent-child hierarchy is structured in a way that is not evident from the graphical representation of the menu structure.


Parent-child hierarchy
Corresponding menu structure
Dialog
MB-1
S1
S2
MI-2
MI-1
MI-1.1
MI-1.2
MI-2.1
MI-2.2
...
S1
MB-1
MI-1
MI-2
MI-1.1
MI-1.2
MI-2.1
MI-2.2
...
S2
is parent of
MENU-HANDLE attribute contains submenu handle value
Sx: submenu control; MB: menu bar control; MI-x.x: menu item

In the above diagram, the first child of the dialog would be the menu-bar control. Its successor would be submenu control S1, and so on. To go from menu item MI-1 to submenu S1, you query for the MENU-HANDLE attribute value of MI-1. The value you get is the handle value of S1.

Creating a Toolbar

There are two ways of creating toolbars and their items:

- Use the dialog editor; or
- use Natural code to create them dynamically.

To use the dialog editor

1. Double-click on the toolbar or from the Natural Menu, select "Dialog > Toolbar".
   The toolbar attributes window opens.
2. Add toolbar items by clicking on the "New" push button.
3. Assign bitmap file names and other attribute values to the new toolbar item.

If you want to use Natural code for dynamic creation, you can study how to build a tool bar in code. Use the enhanced dialog list mode to list a dialog with an editor-built tool bar.

Sharing Menu Structures, Toolbars and DILs (MDI Application)

An MDI (multiple document interface) application consists of a frame dialog that provides the menu structure, toolbar, and DIL shared among all child dialogs. An MDI frame dialog allows you to tile or cascade its child dialogs.

Note: You may only share the toolbar if the PARENT of the toolbar is the dialog of the highest level (the main dialog of an application).

To create an MDI frame dialog

1. Use the dialog editor, and go to the dialog object’s attributes window.
2. Choose "MDI frame window" in the "Type" entry.

An MDI frame dialog must not contain dialog elements other than menu-bar control, submenu control, menu item, toolbar, and toolbar item.

To create an MDI child dialog

1. Use the dialog editor, and go to the dialog object’s attributes window.
2. Choose "MDI child window" in the "Type" entry.

An MDI child dialog:

- can be moved and sized only inside the area of their MDI frame dialog;
- can be maximized to the full size of the area of their MDI frame dialog;
- can be minimized, after which its icon appears at the bottom of its MDI frame dialog;
- can have its own menu structure, toolbar, and DIL. Those do not appear inside the child dialog but are displayed in the MDI frame dialog when the child dialog is active. When another MDI child dialog becomes active, the menu structure, toolbar, and DIL change at the same time;
- can be arranged in a tile or cascade by setting a menu item’s attribute MENU-ITEM-TYPE to the values "MDI Cascade" or "MDI Tile";
- can have its title added to the end of an MDI-WINDOWMENU type submenu control. By choosing one of these menu items, the corresponding MDI child dialog becomes active.

If you want to open an MDI child dialog from within an MDI frame dialog, you can, for example, create a menu item in a menu structure of an MDI frame dialog and define a click event for the menu item. You then write the OPEN DIALOG code for opening an MDI child dialog in the click event handler. The end user will open the MDI child dialog from within the MDI frame dialog by clicking on the menu item, triggering the click event handler.

Example:

```
OPEN DIALOG ’MDICHILD’ #DLG$WINDOW #CHILD-ID
```

The first operand is the name of the dialog created by the dialog editor by selecting "MDI child window" in the "Type" selection box. The second operand is the parent of the new MDI child dialog. This must be the MDI frame dialog. The third operand is a Natural variable defined as I4 in the dialog’s data areas. This variable receives the dialog ID returned by the system.

Note: #DLG$WINDOW is a generated variable.

You can also open an MDI child dialog from within another MDI child dialog (open a sibling of your MDI child dialog). Then you write a similar click-event handler as above:

Example:

```
OPEN DIALOG ’MDICHILD’ #DLG$PARENT #CHILD-ID
```

The first and the third operands are the same as above. The second operand must be the parent of both MDI child dialogs.

Note: #DLG$PARENT is a generated variable.

Executing Standardized Procedures

For procedures frequently needed in event-driven applications, the following is available:

- a set of PROCESS GUI statement actions and
- a set of NGU-prefixed subprograms and dialogs in library SYSTEM.

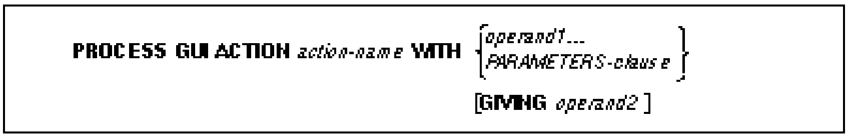
Examples for frequently needed procedures are starting up a message box, reading the lines entered into an edit area control, or dynamically creating dialog elements.

For your convenience, the local data areas NGULKEY1 and NGULFCT1 are automatically included in the list of local data areas used by any new dialog.

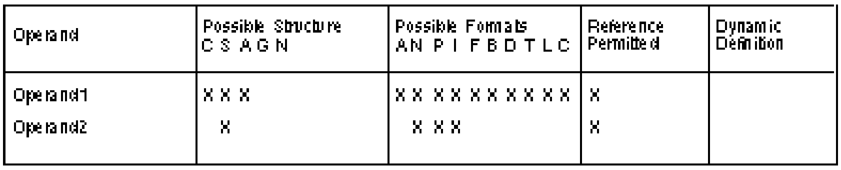
- NGULFCT1 is necessary to use the NGU-prefixed subprograms and dialogs;
- NGULKEY1 lists reserved keywords to be used in any event-handler code. This enables you to refer to certain attribute values by the more meaningful keyword rather than by the numeric IDs. It also enables you to use meaningful dialog element names as parameters.

For more information on the PROCESS GUI statement actions, subprograms and dialogs available, and on the parameters that can be passed, see the chapter Executing Standardized Procedures of the Natural Dialog Components Manual.

PROCESS GUI Statement



```
PROCESS GUI ACTION action-name WITH {operand1...          }
                                    {PARAMETERS-clause    }
                                    [GIVING operand2 ]
```



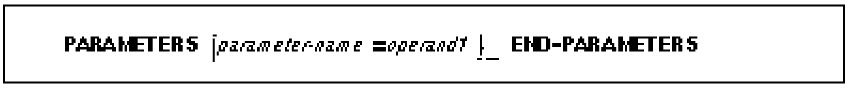
Operand	Possible Structure					Possible Formats										Reference Permitted	Dynamic Definition
	C	S	A	G	N	A	N	P	I	F	B	D	T	L	C		
Operand1	X	X	X			X	X	X	X	X	X	X	X	X	X	X	
Operand2		X					X	X	X							X	

The PROCESS GUI statement is used to perform an action. An action in this context is a procedure frequently needed in event-driven applications.

As *action-name*, you specify the name of the action to be invoked.

As *operand1*, you specify the parameter(s) to be passed to the action. The parameters are passed in the sequence in which they are specified.

For the action "ADD", you can also pass parameters by name (instead of position); to do so, you use the *PARAMETERS-clause*:



```
PARAMETERS parameter-name =operand1 ... END-PARAMETERS
```

This clause can only be used for the action "ADD", not for any other action.

As *operand2*, you can specify a field to receive the response code from the invoked action after the action has been performed.

Linking Dialog Elements to Natural Variables

In cases where you want to map database fields or other program variables to the user interface, input-field controls and selection-box controls may be linked to Natural variables. This makes it easier to modify and query them.

If the end user has entered data in an input-field control or a sebox control and sets the focus to another dialog element, a leave event occurs and the content (STRING) is moved to the variable. Thus, the variable is updated. Note that the variable will *not* be updated if the end user enters data and a change event occurs.

To refresh the content of the dialog element after the linked variable has been modified in code

Use the PROCESS GUI statement action REFRESH-LINKS.

Modifying and querying input-field controls with the ASSIGN statement would normally work like this:

Example:

```
...
#IF-1.STRING := ’12345’
#TEXT := #IF-1.STRING
...
```

However, you can also link a Natural variable to the input-field control or selection box control. You can also link an indexed variable to a dialog element or an array of dialog elements.

To link a variable in Natural code, set the attribute LINKED to TRUE and modify the attribute VARIABLE by setting it to the Natural variable name:

Example:

```
...
#IF-1.LINKED := TRUE
#IF-1.VARIABLE := MYVARIABLE
...
```

To use the dialog editor to enter the name of the Natural variable

1. Double-click on your input-field control.
   The corresponding attributes window appears.
2. Click on the "Source" push button to the right of the "String" entry.
   The "Source for *handlename*" dialog box appears.
3. Choose "Linked variable".
4. Enter the variable name (such as MYVARIABLE in the example above).

There are two possibilities to link an indexed variable such as "MYVARIABLE (A20/1:5)":

- you link a single dialog element to the indexed variable; then you specify the index, such as "MYVARIABLE(2)" in the variable name field of the "Source for *handlename*" dialog box, or
- you link an array of dialog elements to the indexed variable; then you do not specify an index in the variable name field. In this case, the occurrences of the array and the index of the variable must be compatible. "MYVARIABLE (A20/1:5)" could be linked to a one-dimensional array with up to five occurrences.

Validating Input in a Dialog Element

If an input-field control or a sebox control is linked to a Natural variable, this dialog element may be checked automatically when it loses the focus to another dialog element in the same dialog. This enables you to validate the end user’s input. An input field control or a sebox control will not be checked when the end user clicks on a menu item or switches to another application.

If you specify an edit mask with one of these two dialog elements, the field content is checked against this edit mask plus the Natural data type of the linked variable.

If no edit mask is specified, the field content is checked against the Natural data type only.

There are two ways of specifying an edit mask in an input-field control or a selection box control:

- Use Natural code; or
- use the dialog editor.

The Natural code might look like this:

Examples:

```
...
/* Create an input-field control
   1 #IF-1 HANDLE OF INPUTFIELD
...
/* Assign the Edit Mask
#IF-1.EDIT-MASK := ’999’
```

To specify the edit mask with the dialog editor

Open the input-field control’s attribute window and use the "Edit Mask" entry.

When the field check fails, a message box comes up where the end user can choose "Retry" or "Cancel". "Retry" means that the entered text string remains unchanged and can be corrected. "Cancel" means that the field is reset to the current content of the linked variable.

Storing and Retrieving Client Data for a Dialog Element

For a number of dialog elements, the CLIENT-DATA attribute may hold an arbitrary I4 value. This may be useful for linking data to a specific dialog element. A list-box item, for example, can receive and pass on the ISN of a database record. The CLIENT-DATA attribute value may be changed at any time.

In Natural code, this might look like this:

Example:

```
DEFINE DATA
LOCAL
  1 #LBITEM-1 HANDLE OF LISTBOXITEM

  1 #ISN (I4)
    ...
END-DEFINE
...
READ...
   #LBITEM-1.CLIENT-DATA:= #ISN
END-READ
...
```

Note: The CLIENT-DATA attribute of a dialog is reserved for its dialog ID.

Client data may also be set and retrieved as alphanumeric string. In this case, you use the CLIENT-KEY and CLIENT-VALUE attributes in combination.

1. You first assign a value to the CLIENT-KEY attribute. This determines the key under which the string is to be stored for a dialog element.
2. You then assign an alphanumeric string to the CLIENT-VALUE attribute of the dialog element.

This enables you to store a number of key/value pairs for one dialog element.

Example:

```
#LB-1.CLIENT-KEY:= ’ANYKEY’
#LB-1.CLIENT-VALUE:= ’ANYSTRING’        /* The string to be stored
```

To query a dialog element for a particular string

1. You first assign a CLIENT-KEY value to the dialog element.
2. Then you query the dialog element for the corresponding CLIENT-VALUE.

If you assign a value to the CLIENT-KEY of a dialog element, this value is also valid for subsequent querying and modifying of other dialog elements.

If you query the CLIENT-VALUE of a CLIENT-KEY and there is no such pair among the key/value pairs of the dialog element, an empty string (’ ’) is returned.

It is advisable to reuse keys that are not needed because you may use only a limited number of keys.

Example:

```
#LB-1.CLIENT-KEY:= ’ANYKEY’
IF #LB-1.CLIENT-VALUE EQ ’ANYSTRING’ THEN
...
END-IF
```

Creating Dialog Elements on a Canvas Control

You can use a canvas control as a background to draw the following dialog elements on it: the rectangle, line and graphictext controls. These dialog elements "visualize" information. You can, for example, create three or four rectangle controls, fill them with color and change their size at runtime. This way, you can build your own bar chart.

Once you have created a canvas control in the dialog, you can go on to create the rectangle, line and graphictext controls in it.

Note: Graphictext controls do not repaint the background of the rectangle in which they are located. The background of the rectangle is specified at creation time of the graphictext control. What they do repaint is only the text specified in the text attribute.

▶ To create dialog elements on a canvas control

Use the PROCESS GUI statement action ADD.

The rectangle, line and graphictext controls are then displayed inside the borders of the canvas control; if they exceed the canvas borders, they are clipped.

The following attributes are useful for controlling the behavior of the canvas control and the dialog elements on it:

- OFFSET-X and OFFSET-Y determine the x and y axis offset of the canvas control’s upper border against the upper border of the area by which the rectangle, line or graphictext control have exceeded the canvas control’s borders.
- RECTANGLE-X, RECTANGLE-Y, RECTANGLE-W and RECTANGLE-H determine the size of a rectangle control and its position relative to the underlying canvas control.
- P1-X, P1-Y, P2-X and P2-Y determine the start position (P1*xx*) and the end position (P2*xx*) of a line control relative to the underlying canvas control.

The following example illustrates how to create a canvas control and how to add four rectangle controls and two line controls dynamically (these could be used as a bar chart).

Example:

```
/* In the dialog's local data area, the following must be defined:
01 #CNV1 HANDLE OF CANVAS
01 #XAX HANDLE OF LINE
01 #YAX HANDLE OF LINE
01 #H1 HANDLE OF RECTANGLE
01 #H2 HANDLE OF RECTANGLE
01 #H3 HANDLE OF RECTANGLE
01 #H4 HANDLE OF RECTANGLE
01 #RESPONSE (I4)
/* In the dialog's AFTER-OPEN event handler, the following must be defined:
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #DLG$WINDOW
  TYPE = CANVAS
  HANDLE-VARIABLE = #CNV1
  RECTANGLE-X = 20
  RECTANGLE-Y = 20
  RECTANGLE-W = 200
  RECTANGLE-H = 200
  STYLE = 'F'
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #CNV1
  TYPE = LINE
  HANDLE-VARIABLE = #YAX
  STYLE = 'S'
  P1-X = 20
  P1-Y = 20
  P2-X = 20
  P2-Y = 180
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #CNV1
  TYPE = LINE
  HANDLE-VARIABLE = #XAX
  P1-X = 180
  P1-Y = 180
  P2-X = 20
  P2-Y = 180
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #CNV1
  TYPE = RECTANGLE
  HANDLE-VARIABLE = #H1
  RECTANGLE-X = 20
  RECTANGLE-Y = 180
  RECTANGLE-H = 20
  RECTANGLE-W = -60
  FOREGROUND-COLOUR-NAME = BLACK
  BACKGROUND-COLOUR-NAME = RED
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #CNV1
  TYPE = RECTANGLE
```

```
  HANDLE-VARIABLE = #H2
  RECTANGLE-X = 40
  RECTANGLE-Y = 180
  RECTANGLE-H = 20
  RECTANGLE-W = -40
  FOREGROUND-COLOUR-NAME = BLACK
  BACKGROUND-COLOUR-NAME = BLUE
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH PARAMETERS
  PARENT = #CNV1
  TYPE = RECTANGLE
  HANDLE-VARIABLE = #H3
  RECTANGLE-X = 60
  RECTANGLE-Y = 180
  RECTANGLE-H = 20
  RECTANGLE-W = -55
  FOREGROUND-COLOUR-NAME = BLACK
  BACKGROUND-COLOUR-NAME = GREEN
END-PARAMETERS
GIVING RESPONSE
PROCESS GUI ACTION ADD WITH
PARAMETERS
  PARENT = #CNV1
  TYPE = RECTANGLE
  HANDLE-VARIABLE = #H4
  RECTANGLE-X = 80
  RECTANGLE-Y = 180
  RECTANGLE-H = 20
  RECTANGLE-W = -80
  FOREGROUND-COLOUR-NAME = BLACK
  BACKGROUND-COLOUR-NAME = MAGENTA
END-PARAMETERS
GIVING RESPONSE
```

Working with ActiveX Controls

ActiveX controls are third-party custom controls that you can integrate in a Natural dialog.

Terminology

ActiveX controls and Natural use different terminology in two cases:

ActiveX Control	Natural
Property	Attribute
Method	PROCESS GUI Statement Action

How To Define an ActiveX Control

The handle of an ActiveX Control is defined similar as a built-in dialog element, but its individual aspects are coded in double angle brackets.

Example:

```
01    #OCX-1 HANDLE OF <<OCX-Table.TableCtrl.1 [Table Control]>>
```

In the above example, ’Table.TableCtrl.1’ is the program ID (ProgID) under which the ActiveX control is registered in the system registry. The prefix ’OCX-’ identifies the control as an ActiveX control. ’[Table Control]’ is an optional part of the definition and provides a readable name.

How To Create an ActiveX Control

You create an instance of an ActiveX control by using the PROCESS GUI statement action ADD. To do so, the value of the TYPE attribute must be the ActiveX control’s ProgID prefixed with the string ’OCX-’ and put in double angle brackets. The ProgID is the name under which the control is registered in the system registry. You can additionally provide a readable name in square brackets. In addition to that, you can set Natural attributes such as RECTANGLE-X as well as the ActiveX control’s properties. The property name must be preceded by the string ’PROPERTY-’ and this combination must be put in double angle brackets. Furthermore, you can suppress the ActiveX control’s events. To do this, the event name must be preceded by the string ’SUPPRESS-EVENT’ this combination must be delimited by double angle brackets. The value of the SUPPRESS-EVENT property is either the Natural keyword ’SUPPRESSED’ or ’NOT-SUPPRESSED’.

Example:

```
PROCESS GUI ACTION ADD
    WITH PARAMETERS
       HANDLE-VARIABLE = #OCX-1
       TYPE = <<OCX-Table.TableCtrl.1 [Table Control]>>
       PARENT = #DLG$WINDOW
       RECTANGLE-X = 44
       RECTANGLE-Y = 31
       RECTANGLE-W = 103
       RECTANGLE-H = 46
       <<PROPERTY-HeaderColor>> = H'FF0080'
       <<PROPERTY-Rows>> = 16
       <<PROPERTY-Columns>> = 4
       <<SUPPRESS-EVENT-RowMoved>> = SUPPRESSED
       <<SUPPRESS-EVENT-ColMoved>> = SUPPRESSED
    END-PARAMETERS
```

Accessing Simple Properties

Simple properties are properties that do not have parameters. Simple properties of an ActiveX control are addressed like attributes of built-in controls. The attribute name is built by prefixing the property name with 'PROPERTY-' and enclosing it in angle brackets.The properties of an ActiveX control are displayed in the Component Browser. The following examples assume that the ActiveX control #OCX-1 has the simple properties 'CurrentRow' and 'CurrentCol'.

Example:

```
 * Get the value of a property.
 #MYROW := #OCX-1.<&ltPROPERTY-CurrentRow>>
 * Put the value of a property.
 #OCX-1.<&ltPROPERTY-CurrentCol>> := 17
```

The data types of ActiveX control properties are those defined by OLE Automation. In Natural, each of these data types is mapped to a corresponding Natural data type. The following table shows which OLE Automation data type is mapped to which Natural data type.

OLE Automation data type	NATURAL data type
VT_BOOL	L
VT_BSTR	A dynamic
VT_CY	P15.4
VT_DATE	T
VT_DECIMAL	Pn.m
VT_DISPATCH	HANDLE OF OBJECT
VT_ERROR	I4
VT_I1	I2
VT_I2	I2
VT_I4	I4
VT_INT	I4
VT_R4	F4
VT_R8	F8
VT_U1	B1
VT_U2	B2
VT_U4	B4
VT_UINT	B4
VT_UNKNOWN	HANDLE OF OBJECT
VT_VARIANT	(any Natural data type)
OLE_COLOR (VT_UI4)	B3
VT_FONT (VT_DISPATCH IFontDisp*)	HANDLE OF FONT, HANDLE OF OBJECT (IFontDisp*) A dynamic
VT_PICTURE (VT_DISPATCH IPictureDisp*)	HANDLE OF OBJECT (IPictureDisp*) A dynamic

Read the table in the following way: Assume an ActiveX control #OCX-1 has a property named ’Size’, which is of type VT_R8. Then the expression #OCX-1.<<PROPERTY-SIZE>> has the type F8 in Natural.

Note: The Component Browser displays the corresponding Natural data types directly.

Some special data types are considered individually in the following:

Colors

A property of type Color appears in Natural as a B3 value. The B3 value is interpreted as an RGB color value. The three bytes contain the red, green and blue elements of the color, respectively. Thus for example H’FF0000’ corresponds to red, H’00FF00’ corresponds to green, H’0000FF’ corresponds to blue and so on.

Example:

```
...
 01 #COLOR-RED (B3)
...
 #COLOR-RED := H’FF0000’
 #OCX-1.<<PROPERTY-BackColor>> := #COLOR-RED
...
```

Pictures

A property of type Picture appears in Natural as HANDLE OF OBJECT. Alternatively you can assign an Alpha value to a Picture property. The Alpha value must then contain the file name of a Bitmap (.bmp) file.

Example (usage of Picture properties):

```
...
 01 #MYPICTURE HANDLE OF OBJECT
...
 * Assign a Bitmap file name to a Picture property.
 #OCX-1.<<PROPERTY-Picture>>:= ’11100102.bmp’
 *
 * Get it back as an object handle.
 #MYPICTURE := #OCX-1.<<PROPERTY-Picture>>
 *
 * Assign the object handle to a Picture property of another control.
 #OCX-2.<<PROPERTY-Picture>>:= #MYPICTURE
...
```

Fonts

A property of type Font appears in Natural as HANDLE OF OBJECT. You can alternatively assign a HANDLE OF FONT to a Font property. Additionally you can assign an Alpha value to a Font property. The Alpha value must then contain a font specification in the form that is returned by the STRING attribute of a HANDLE OF FONT.

Example 1 (using HANDLE OF OBJECT):

```
...
01 #MYFONT HANDLE OF OBJECT
...
* Create a Font object.
CREATE OBJECT #MYFONT OF CLASS ’StdFont’
#MYFONT.Name := ’Wingdings’
#MYFONT.Size := 20
#MYFONT.Bold := TRUE
*
* Assign the Font object as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #MYFONT
...
```

Example 2 (using HANDLE OF FONT):

```
...
01 #FONT-TAHOMA-BOLD-2 HANDLE OF FONT
...
* Create a Font handle.
PROCESS GUI ACTION ADD WITH PARAMETERS
    HANDLE-VARIABLE = #FONT-TAHOMA-BOLD-2
    TYPE = FONT
    PARENT = #DLG$WINDOW
    STRING = ’/Tahoma/Bold/0 x -27/ANSI VARIABLE SWISS DRAFT/W/2/3/’
END-PARAMETERS GIVING *ERROR
...
* Assign the Font handle as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #FONT-TAHOMA-BOLD-2
...
```

Example 3 (using a font specification string):

```
...
01 #FONT-TAHOMA-BOLD-2 HANDLE OF FONT
...
* Create a Font handle.
PROCESS GUI ACTION ADD WITH  PARAMETERS
    HANDLE-VARIABLE = #FONT-TAHOMA-BOLD-2
    TYPE = FONT
    PARENT = #DLG$WINDOW
    STRING = ’/Tahoma/Bold/0 x -27/ANSI VARIABLE SWISS DRAFT/W/2/3/’
END-PARAMETERS GIVING *ERROR
...
* Assign the font specification as value to a Font property.
#OCX-1.<<PROPERTY-TitleFont>> := #FONT-TAHOMA-BOLD-2.STRING
...
```

Variants

A property of type Variant is compatible with any Natural data type. This means that the type of the expression #OCX-1.<<PROPERTY-Value>> is not checked by the compiler, if "Value" is a property of type Variant. So the assignments #OCX-1.<<PROPERTY-Value >> := #MYVAL and #MYVAL := #OCX-1.<<PROPERTY-Value >> are allowed independently of the type of the variable #MYVAL. It is however up to the ActiveX control to accept or reject a particular property value at runtime, or to deliver the value in the requested format. If it does not, the ActiveX control will usually raise an exception. This exception is returned as a Natural error code to the Natural program. Here it can be handled in the usual way in an ON ERROR block. You should check the documentation of the ActiveX control to find out which data formats are actually allowed for a particular property of type Variant.

An expression like #OCX-1.<<PROPERTY-Value>> (where "Value" is a Variant property) can occur as source operand in any statement. However, it can be used as target operand only in assignment statements.

Examples (usage of Variant properties):
(Assume that ’Value’ is a property of type Variant of the ActiveX control #OCX-1)

```
...
01 #STR1 (A100)
01 #STR2 (A100)
...
* These statements are allowed, because the Variant property is used
* as source operand (its value is read).
#STR1 := #OCX-1.<<PROPERTY-Value>>
COMPRESS #OCX-1.<<PROPERTY-Value>> ’XYZ’ to #STR2
...
* This leads to an error at compiletime, because the Variant
* property is used as target operand (its value is modified) in
* a statement other than an assignment.
COMPRESS #STR1 "XYZ" to #OCX-1.<<PROPERTY-Value>>
...
* This statement is allowed, because the Variant property is used
* as target operand in an assignment.
COMPRESS #STR1 ’XYZ’ to #STR2
#OCX-1.<<PROPERTY-Value>> := #STR2
...
```

Arrays

A property of type SAFEARRAY of up to three dimensions appears in a Natural program as an array with the same dimension count, occurrence count per dimension and the corresponding format. (Properties of type SAFEARRAY with more than three dimensions cannot be used in Natural programs.) The dimension and occurrence count of an array property is not determined at compiletime but only at runtime. This is because this information is variable and is not defined at compiletime. The format however is checked at compiletime.

Array properties are always accessed as a whole. So no index notation is necessary and allowed with an array property.

Examples (usage of Array properties):
(Assume that ’Values’ is a property of the ActiveX control #OCX-1 an has the type SAFEARRAY of VT_I4)

```
...
01 #VAL-L (L/1:10)
01 #VAL-I (I4/1:10)
...
* This statement is allowed, because the format of the property
* is data transfer compatible with the format of the receiving array.
* However, if it turns out at runtime that the dimension count or
* occurrence count per dimension do not match, a runtime error will
* occur.
VAL-I(*) := #OCX-1.<<PROPERTY-Values>>
...
* This statement leads to an error at compiletime, because
* the format of the property is not data transfer compatible with
* the format of the receiving array.
VAL-L(*) := #OCX-1.<<PROPERTY-Values>>
...
```

Using The PROCESS GUI Statement

The methods of ActiveX controls are called as actions in a PROCESS GUI statement. The same is the case with the complex properties of ActiveX controls (i. e. properties that have parameters). The methods and properties of an ActiveX control are displayed in the Component Browser

Performing Methods

To perform a method of an ActiveX control the PROCESS GUI statement is used. The name of the corresponding PROCESS GUI action is built by prefixing the method name with ’METHOD-’ and enclosing it in angle brackets. The ActiveX control handle and the method parameters (if any) are passed in the WITH clause of the PROCESS GUI statement The return value of the method (if any) is received in the variable specified in the USING clause of the PROCESS GUI statement.

This means: To perform a method, you enter a statement

```
    PROCESS GUI ACTION <<METHOD-methodname>> WITH handlename [ parameter]...
[USING method-return-operand]..
```

Examples:

```
* Performing a method without parameters:
PROCESS GUI ACTION <<METHOD-AboutBox>> WITH #OCX-1
* Performing a method with parameters:
PROCESS GUI ACTION <<METHOD-CreateItem>> WITH #OCX-1 #ROW #COL #TEXT
* Performing a method with parameters and a return value:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN
```

Formats and length of the method parameters and the return value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser

Getting Property Values

To get the value of a property that has parameters, the name of the corresponding PROCESS GUI action is built by prefixing the property name with ’GET-PROPERTY-’ and enclosing it in angle brackets. The ActiveX control handle and the property parameters (if any) are passed in the WITH clause of the PROCESS GUI statement The property value is received in the USING clause of the PROCESS GUI statement.
This means:
To get the value of a property that has parameters, you enter a statement

```
    PROCESS GUI ACTION <<GET-PROPERTY-propertyname>> WITH handlename [ parameter]
... USING get-property-operand
```

Example:

```
PROCESS GUI ACTION <<GET-PROPERTY-ItemHeight>> WITH #OCX-1 #ROW #COL USING #ITEMHEIGHT
```

Formats and length of the property parameters and the property value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser

Putting Property Values

To put the value of a property that has parameters, the name of the corresponding PROCESS GUI action is built by prefixing the property name with ’PUT-PROPERTY-’ and enclosing it in angle brackets. The ActiveX control handle and the property parameters (if any) are passed in the WITH clause of the PROCESS GUI statement The property value is passed in the USING clause of the PROCESS GUI statement.

This means:
To put the value of a property that has parameters, you enter a statement

```
    PROCESS GUI ACTION <<PUT-PROPERTY-propertyname>> WITH handlename [ parameter]
... USING put-property-operand
```

Example:

```
    PROCESS GUI ACTION <<PUT-PROPERTY-ItemHeight>> WITH #OCX-1 #ROW #COL USING #ITEMHEIGHT
```

Formats and length of the property parameters and the property value are checked at compiletime against the definition of the method, as it is displayed in the Component Browser

Optional Parameters

Methods of ActiveX controls can have optional parameters. This is also true for parameterized properties. Optional parameters need not to be specified when the method is called. To omit an optional parameter, use the placeholder 1X in the PROCESS GUI statement. To omit n optional parameters, use the placeholder nX.

In the following example it is assumed that the method SetAddress of the ActiveX control #OCX-1 has the parameters FirstName, MiddleInitial, LastName, Street and City, where MiddleInitial, Street and City are optional. Then the following statements are correct:

Example:

```
* Specifying all parameters.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName Street City
* Omitting one optional parameter.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName 1X LastName Street City
* Omitting the optional parameters at end explicitly.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName 2X
* Omitting the optional parameters at end implicitly.
PROCESS GUI ACTION <<METHOD-SetAddress>> WITH #OCX-1
FirstName MiddleInitial LastName
```

Omitting a non-optional (mandatory) parameter results in a syntax error.

Error handling

The GIVING clause of the PROCESS GUI statement can be used as usual to handle error conditions. The error code can either be caught in a user variable and then be handled, or the normal Natural error handling can be triggered and the error condition be handled in an ON ERROR block.

Example:

```
DEFINE DATA LOCAL
1 #RESULT-CODE (N7)
...
END-DEFINE
...
* Catching the error code in a user variable:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN GIVING #RESULT-CODE
*
* Triggering the Natural error handling:
PROCESS GUI ACTION <<METHOD-RemoveItem>> WITH #OCX-1 #ROW #COL USING #RETURN GIVING *ERROR-NR
...
```

Special error conditions that can occur during the execution of ActiveX control methods are:

- A method parameter, method return value or property value could not be converted to the data format expected by the ActiveX control. (These format checks are normally already done at compiletime. In these cases no runtime error can be expected. However, note that method parameters, method return values or property values defined as Variant are not checked at compiletime. This applies also for arrays and for those data types that can be mapped to several possible Natural data types.)
- A COM or Automation error occurs while locating and executing a method.
- The ActiveX control raises an exception during the execution of a method.

In these cases the error message contains further information provided by the ActiveX control, which can be used to determine the reason of the error with the help of the documentation of the ActiveX control.

Using Events With Parameters

Events sent by ActiveX controls can have parameters. In the controls event-handler sections, these parameters can be queried. Parameters passed by reference can also be modified. The events of an ActiveX control, the names and data types of the parameters and the fact if a parameter is passed by value or by reference is all displayed in the Component Browser.

Event parameters of an ActiveX control are addressed like attributes of built-in controls. The attribute name is built by prefixing the parameter name with ’PARAMETER-’ and enclosing it in angle brackets. Alternatively, parameters can be addressed by position. This means the attribute name is built by prefixing the number of the parameter with ’PARAMETER-’ and enclosing it in angle brackets.The first parameter of an event has the number 1, the second the number 2 and so on. These attribute names are only valid inside the event handler of that particular event.

In the following examples it is assumed that a particular event of the ActiveX control #OCX-1 has the parameters KeyCode and Cancel. Then the event handler of that event might contain the following statements:

Example:

```
* Querying a parameter by name:
#PRESSEDKEY := #OCX-1.<<PARAMETER-KeyCode>>
* Querying a parameter by position:
#PRESSEDKEY := #OCX-1.<<PARAMETER-1>>
```

Parameters that are passed by reference can be modified in the event handler. In the following example it is assumed that the Cancel parameter is passed by reference and is thus modifiable. Then the event handler might contain the following statements:

Example:

```
* Modifying a parameter by name:
#OCX-1.<<PARAMETER-Cancel>>:= TRUE
* Modifying a parameter by position:
#OCX-1.<<PARAMETER-2>>:= TRUE
```

Suppressing Events At Runtime

To suppress or unsuppress an event of an ActiveX control at runtime, modify the corresponding suppress event attribute of the control. The name of the suppress event attribute is built by prefixing the event name with ’SUPPRESS-EVENT-’ and enclosing it in angle brackets. The events of an ActiveX control are displayed in the Component Browser.

The following example assumes that the ActiveX control #OCX-1 has the event ColMoved.

Example:

```
* Suppress the event.
#OCX-1.<<SUPPRESS-EVENT-ColMoved>> := SUPPRESSED
* Unsuppress the event.
#OCX-1.<<SUPPRESS-EVENT-ColMoved>> := NOT-SUPPRESSED
```

Working with Arrays of Dialog Elements

It is sometimes convenient to arrange dialog elements in one or two dimensions. If, for example, you want to arrange several radio-button controls in one column, it is possible to draw the first one and specify the others as a one-dimensional array.

To work with arrays of dialog elements:

1. Click the "Array" button in the radio-button control’s attributes window.
   The "Array Specification" dialog box appears.
2. Enter:

- the number of dimensions;
- the bounds of the first and second dimension, if applicable;
- the spacing on the x and y axis in pixels (depending on whether the array is arranged in rows or in columns);
- the arrangement (rows or columns).

The array will now be treated as a graphical entity. Note that you will have to assign a common GROUP-ID attribute to each radio-button control. This will enable you to treat the array as a logical entity.

For each dialog element in an array, the following attributes may be specified separately:

- STRING
- DIL-TEXT
- BITMAP-FILE-NAME

In an event handler for an array of dialog elements, the system variable *CONTROL will denote one of the array elements.

If a variable is selected as the source of an attribute value, the array must contain at least the index ranges of the dialog element.

If a message file ID is specified as the source of an attribute value, consecutive messages are taken for the array’s sequence of dialog elements.

In an array of dialog elements, you can assign one value to all dialog elements in the array using the (*) notation or a range, such as in the following examples:

Examples:

```
#PB-1.ENABLED(*) := TRUE    /*invalid
#PB-1.ENABLED(1:3) := TRUE  /*invalid
```

An alternative way of creating a sequence of identical dialog elements is to duplicate or copy and paste an individual dialog element and use the grid plus the cross-hair cursor to place them.

The following example illustrates how to set the STRING attribute of occurence 2 in a one-dimensional push-button array:

Examples:

```
#PB-2.STRING(2) := ’HUGO’
```

Working with Control Boxes

A control box is is used to enhance the effectiveness of the nested control support. However, control boxes have a number of unique features that merit their separate discussion.

Control boxes are, in themselves, fairly inert controls, belonging to the same category as text constants and group frames in that they cannot receive the focus and do not receive any mouse or keyboard input. Instead, they are intended to act as general-purpose containers for other controls (including, possibly, other control boxes), in order to build up a control hierarchy. In doing so, control boxes support three styles which are worthy of special mention here:

- Because it is often desirable to be able to group controls together for convenience, but not desirable that the user actually sees the container itself, control boxes can be marked with the style ’transparent’. In this case, no parts of the control box are drawn, and any underlying colors and controls show through.
- Control boxes can also be marked with the style ’exclusive’. When an exclusive control box is made visible, either in the dialog editor or at runtime, all other sibling control boxes that are also marked as ’exclusive’ are hidden. This applies to edit-time and runtime in a slightly different way. At runtime, setting the VISIBLE attribute of an exclusive control box to TRUE hides all its exclusive siblings and sets their VISIBLE attribute to FALSE. At edit-time, whenever an exclusive control box or one of its descendants is selected, the exclusive control box becomes visible and all other exclusive siblings are hidden. However, in this latter case the VISIBLE attribute of the controls concerned is unaffected. This implies that the exclusive control box that is initially visible when the dialog is run is independent of the exclusive control box that was visible at the time the dialog was last saved.
- Additionally, control boxes support the ’size to parent’ style. When a container control, or the dialog itself, is resized, all child control boxes (if any) with this style set are resized to entirely fill the parent’s client area. The same applies when this style is first set in the dialog editor. However, it is still possible to resize such control boxes independently of their container.

Purpose of exclusive control boxes

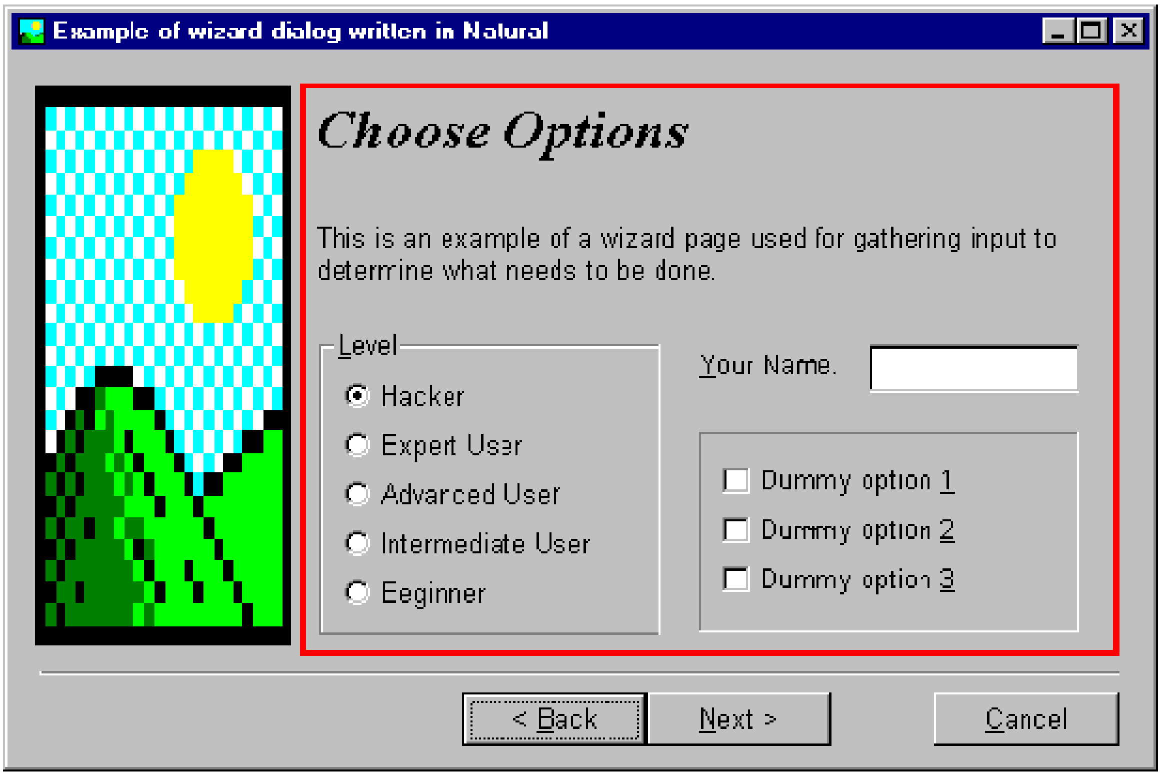
Exclusive control boxes, as described above, are primarily intended for situations where it is necessary to manage several overlapping "pages" of controls occupying the same region of a dialog. Without the auto-hiding feature which exclusive control boxes provide, it would be very difficult indeed for a user to handle this situation in the dialog editor, as many controls would be partially or completely overlapped by others. Of course, one could move the control to the front of the control sequence during editing, but this would be highly inconvenient, and one would have to remember to move the control back before continuing.

Using exclusive control boxes, editing a control in this situation is as simple as selecting it. For controls that are not currently on display, the selection can be made via the combo box in the dialog editor’s status bar or by using the <Tab> key to walk through the controls sequentially until the target control is reached. When a control that is a descendant of an exclusive control box is selected, that exclusive control box is made visible (if not already so), and the previously visible exclusive control box is hidden. These changes have no impact on the generated dialog source code and the runtime state of the dialog.

Examples of use of exclusive control boxes

Although the design of control boxes was intended to keep them as general as possible, two possible situations where overlapping control pages are desired (and hence where exclusive control boxes become extremely useful) are worthy of special mention here:

- Wizard dialogs.
- Tabbed dialogs ("Property sheets").



Example of wizard dialog written in Natural

Choose Options

This is an example of a wizard page used for gathering input to determine what needs to be done.

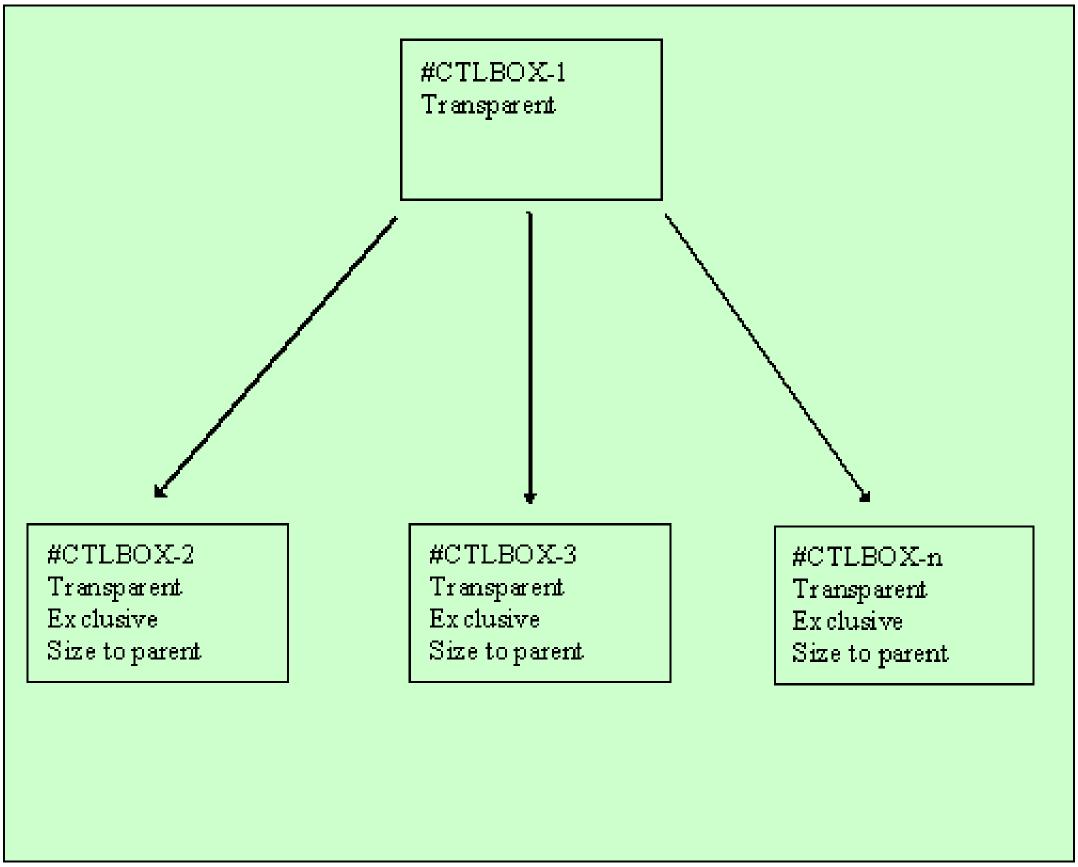
Level

- ● Hacker
- ○ Expert User
- ○ Advanced User
- ○ Intermediate User
- ○ Beginner

Your Name:

- □ Dummy option 1
- □ Dummy option 2
- □ Dummy option 3

Within the rectangle highlighted in red, the so-called "wizard pages" are displayed. Within this area, we use a 2-level hierarchy of control boxes in order to implement the required functionality:



Here, #CTLBOX-1 is used as the "master" control box, which makes resizing of the pages easier later, should this become necessary. Because all child control boxes are marked with the style ’size to parent’, we can resize the wizard page area simply by resizing #CTLBOX-1.

The child control boxes are used to implement the actual wizard pages. #CTLBOX-2 contains the controls used for wizard page 1, #CTLBOX-3 contains the controls for wizard page 2, and so on.

Creation of the wizard pages

Creation of the wizard pages typically involves the following steps:

1. Create the top-level ("master") control box as for any other control.
2. Via its attributes window, set the ’transparent’ style.
3. Create another control box within the first one. The new control box automatically becomes a child of the first one, because control boxes are always containers.
4. Via the attributes window for the child control box, set the ’transparent’, ’exclusive’ and ’size to parent styles’. Because the ’size to parent’ style is set, the child control box expands to fill its container.
5. Now you can start adding the controls onto the newly-created control box, which becomes wizard page 1.
6. Adding a new wizard page is most easily achieved by selecting the child control box you wish to immediately precede the new one, then using the clipboard copy and paste commands. Before doing the copy, Natural will prompt you as to whether you want the child controls to be copied, too. Answer this question with ’No’.
7. Because the newly added child control box also has the exclusive flag set, the previously displayed child control box is hidden, and the new blank one is shown, ready for you to start adding a new set of controls as for the first wizard page.

Switching between the wizard pages at edit-time

Switching between the pages at edit time can be most simply achieved by selecting the child control box for the appropriate page, or one of the controls on it, from the combo box in the dialog editor’s status bar.

Creating the divider line

The divider line between the push buttons and the wizard pages can be implemented as a very thin group box (2 pixels high) with no caption. The still slightly visible sides of the group box at each end can be masked out by using a transparent control box which comes after the group frame in the control sequence. Make sure the ’control clipping’ style for the dialog is switched on for this technique to work.

Implementing the ’Back’ and ’Next’ push buttons

Firstly, define a local variable for the dialog to store the handle of the currently active page. E.g.:

```
 01 #ACTPAGE HANDLE OF CONTROLBOX ...
```

Secondly, set this variable to the handle of the first wizard page in the AFTER-OPEN event for the dialog:

```
 #ACTPAGE := #CTLBOX-1.FIRST-CHILD ..
```

where #CTLBOX-1 is the handle of the top-level control box.
Now we are ready to implement the CLICK event code for the ’Next’ push button (#PB-NEXT). This could look something like this:

```
IF #ACTPAGE.SUCCESSOR = NULL-HANDLE
  CLOSE DIALOG *DIALOG-ID
ELSE
  REPEAT
    #ACTPAGE := #ACTPAGE.SUCCESSOR
    WHILE #ACTPAGE.ENABLED = FALSE
  END-REPEAT
  #ACTPAGE.VISIBLE := TRUE
  IF #ACTPAGE.SUCCESSOR = NULL-HANDLE
    #PB-NEXT.STRING := ’Finish’
    #PB-BACK.ENABLED := FALSE
    #PB-CANCEL.ENABLED := FALSE
  ELSE
    #PB-BACK.ENABLED := TRUE
  END-IF
END-IF
..
```

Note that this logic does not be modified if further wizard pages are added later. Note also that any intermediate wizard pages whose corresponding control box has been disabled are ignored. This allows certain wizard pages to be skipped, based on previous input, by simply setting the relevant control box ENABLED attribute to FALSE. When the last page is reached, the text for the ’Next’ push button is changed to ’Finish’.

The CLICK event code for the ’Back’ push button (#PB-BACK) is very similar:

```
REPEAT
  #ACTPAGE := #ACTPAGE.PREDECESSOR
  WHILE #ACTPAGE.ENABLED = FALSE
END-REPEAT
IF #ACTPAGE.PREDECESSOR = NULL-HANDLE
  #PB-BACK.ENABLED := FALSE
END-IF
#ACTPAGE.VISIBLE := TRUE
..
```

Note that the ’Back’ push button should be initially disabled in the dialog editor.

Clearing all controls on a wizard page

This can be conveniently achieved by selecting any (highest-level) control on the relevant page, then performing a "Select All" from the "Edit" menu to additionally select all the controls siblings. The selected controls can then be deleted as normal.

Example 2 - a tabbed dialog

A tabbed dialog (sometimes called a "property sheet") is very similar in concept to a wizard dialog. The only substantial difference is that instead of navigating between the control "pages" via the ’Next’ and ’Back’ push buttons, the user directly accesses the page he wants by clicking on the appropriate tab. The control page hierarchy can be built up and handled in the dialog editor in the same way as in the wizard dialog example above. Several ActiveX controls are available which provide the actual tabs.

It should be noted, however, that the switching between the pages (i.e., switching between the corresponding control boxes) is not automatic. The Natural programmer must insert code for the ActiveX event raised by a tab switch, find out which tab is selected, and set the VISIBLE attribute of the appropriate (exclusive) control box to TRUE. This cannot be done implicitly by Natural because each ActiveX control can implement its functionality in any way it chooses. There is no standard event raised for a tab switch and no standard method with standard parameters (or standard property) for determining the currently active tab.

An example tabbed dialog, making use of the Microsoft "Tab Strip" ActiveX control (V4-NEST.NS3) is shipped as part of the Natural example libraries.

Working with Error Events

When a runtime error occurs while a dialog is active, the dialog receives an error event. You can specify event-handler code to be executed whenever this error occurs. If no error event-handler code is specified, Natural aborts with an error message and all dialogs will be closed.

You can continue normal dialog processing after error handling by specifying ESCAPE ROUTINE at the end of the event-handler code.

The dialog editor generates an ON ERROR statement for the event handler. If, for example, you want to prevent the end user from closing the entire application when trying to divide an integer by zero, and the parameter ZD is set to ON, the error event-handler code might look like this:

```
COMPRESS ’Natural error’ *ERROR ’occurred.’ INTO #DLG$WINDOW.STATUS-TEXT
    ESCAPE ROUTINE
```

Working with a Group of Radio-Button Controls

radio-button controls are created just like push-button controls or toggle button controls; however, they are grouped using the GROUP-ID attribute. If you define a number of radio-button controls as a group, only one button is selected at any time. The GROUP-ID attribute provides this selection logic.

You group several radio-button controls by assigning them the same GROUP-ID value (group number) in their attributes windows. If the end user clicks on a radio-button control, all other radio-button controls in the dialog with the same GROUP-ID will be deselected. They will also be deselected if one radio-button control is selected by code like the following:

Example:

```
...
   1 #RB-1 HANDLE OF RADIOBUTTON
...
#RB-1.CHECKED := CHECKED  /* Set the CHECKED attribute to value CHECKED
...
```

You also have to bear in mind that the end user should be able to use the keyboard for navigation inside a group of radio-button controls: TAB selects the first radio-button control, and the arrow keys enable you to navigate within the radio-button group. To ensure that Natural automatically allows for such navigation, the radio-button controls must follow each other directly in the navigation sequence. If you are dynamically adding a radio-button control via the PROCESS GUI statement action ADD, this can be achieved by specifying a value for the button’s FOLLOWS attribute.

To edit the navigation sequence

From the menu bar, select "Dialog > Control Sequence".

Working with List-Box Controls and Selection-Box Controls

list-box controls and selection box controls contain a number of items. Both the controls and the items are dialog elements; the controls are the parents of the items.

There are two ways of creating list-box items and sebox items:

- Use Natural code to create individual and multiple list-box items dynamically; or
- use the dialog editor (to add single or arrays of list-box items and sebox items).

In Natural code, this may look like this:

Example:

```
#AMOUNT := 5
ITEM (1) := ’BERLIN’
ITEM (2) := ’PARIS’
ITEM (3) := ’LONDON’
ITEM (4) := ’MILAN’
ITEM (5) := ’MADRID’
PROCESS GUI ACTION ADD-ITEMS WITH #LB-1 #AMOUNT #ITEM (1:5) GIVING #RESPONSE
```

You first specify the number of items you want to create, name the items, and use the PROCESS GUI statement action ADD-ITEMS.

If you want to go through all items of a list-box control to find out which ones are selected, it is advisable to use the SELECTED-SUCCESSOR attribute because if a list-box control contains a large number of items (100, for example), this helps improve performance. If you use SELECTED-SUCCESSOR, you have one query instead of 100 individual queries if you use the attributes SELECTED and SUCCESSOR.

Example:

```
/* Displays the STRING attribute of every SELECTED list-box item
MOVE #LISTBOX.SELECTED-SUCCESSOR TO #LBITEM
REPEAT UNTIL #LBITEM = NULL-HANDLE
   .../* STRING display logic
   MOVE #LBITEM.SELECTED-SUCCESSOR TO #LBITEM
END-REPEAT
```

For performance reasons, you should not use the SELECTED-SUCCESSOR attribute to refer to the same dialog element handle twice, because Natural goes through the list of item handles twice:

Example:

```
/* Displays the STRING attribute of every SELECTED list-box item,
/* but may be slow
MOVE #LISTBOX.SELECTED-SUCCESSOR TO #LBITEM
REPEAT UNTIL #LBITEM = NULL-HANDLE
   IF #LBITEM.SELECTED-SUCCESSOR = NULL-HANDLE /* Searches in the list of items
     IGNORE
   END-IF
   .../* STRING display logic
   MOVE #LBITEM.SELECTED-SUCCESSOR TO #LBITEM /* Searches in the list of items
END-REPEAT                                    /* for the second time
```

To avoid this problem, you use a second variable "#OLDITEM" besides "#LBITEM":

Example:

```
/* Displays the STRING attribute of every SELECTED list-box item
MOVE #LISTBOX.SELECTED-SUCCESSOR TO #LBITEM
REPEAT UNTIL #LBITEM = NULL-HANDLE
   #OLDITEM = #LBITEM
   #LBITEM = #LBITEM.SELECTED-SUCCESSOR/* Searches in the list of items (once)
   IF #LBITEM = NULL-HANDLE
     IGNORE
  END-IF
   .../* Display logic using #OLDITEM.STRING
END-REPEAT
```

If you retrieve the handle values of the selected items, a value other than NULL-HANDLE would normally be returned by selected items. Such a handle value can also be returned by non-selected items if you assign SELECTED-SUCCESSOR a value immediately before retrieving the SELECTED-SUCCESSOR value of a non-selected item, as shown in the following example:

Example:

```
...
PTR := #LB-1.SELECTED-SUCCESSOR
PTR := NOT_SELECTEDHANDLE.SELECTED-SUCCESSOR
IF NOT_SELECTEDHANDLE.SELECTED-SUCCESSOR = NULL-HANDLE THEN
   #DLG$WINDOW.STATUS-TEXT := ’NULL-HANDLE’
ELSE
   COMPRESS ’NEXT SELECTION: ’ PTR.STRING TO #DLG$WINDOW.STATUS-TEXT
END-IF
...
```

If you want to query whether a particular item in a list-box control is selected, you get the best performance by using the SELECTED attribute:

Example:

```
#DLG$WINDOW.STRING:= #LB-1-ITEMS.SELECTED(3)
```

Protecting Selection-Box Controls and Input-Field Controls

To prevent an end user from typing in input data in a sebox control or input-field control, you have several possiblities, for example:

- setting the MODIFIABLE attribute to FALSE for the dialog element, or
- setting session parameter AD=P, or
- using a control variable (CV).

If a sebox control is protected, it is still possible to select items; only values from the item list will be displayed in its input field. If the STRING attribute is set to a value (dynamically or by initialisation) which is not in the item list, the value will not be visible to the end user.

Working with Nested Controls

It is possible to create controls as children of other controls in addition to so-called "top-level" controls, which are direct children of the dialog. Such controls are referred to as nested controls. The parent control is referred to as the container. We will also use the term siblings to refer to a set of child controls which all have the same parent. Clearly, there can be many different sets of sibling controls within a control hierarchy.

Creation of a control hierarchy enables the Natural programmer to group together controls such that they can be manipulated more easily and more efficiently within a Natural program. The following list describes the characteristics of nested controls:

- Their position is relative to the client area of the container control instead of relative to the dialog.
- Their display is clipped to their respective ancestor windows. This means that the areas of the nested control that are outside the boundary of its container are not visible. The dialog editor does not allow dragging of nested controls outside of the container.
- Nested controls are always displayed in front of their container control, regardless of their position in the control sequence.
- Nested controls are moved with their container control. This applies at both edit-time in the dialog editor (when the container is dragged) and at runtime (when the container’s RECTANGLE-X and/or RECTANGLE-Y attributes are modified).
- Nested controls are hidden when the container control is hidden, even though the VISIBLE attribute of the nested control remains unchanged.
- Nested controls are disabled at runtime when the container control is disabled, even though the ENABLED attribute of the nested control remains unchanged and even though the control does not become grayed.
- Nested controls are deleted when the container control is deleted.

Note:
Natural does not impose any arbitrary limits on the number of levels that a control hierarchy may contain. The level number for a particular control is displayed together with the control’s name in the dialog editor status bar combo box.

Which control types can be containers?

Not all control types are capable of acting as a container. It is not possible to create a control as a child of an input field, for example. There are currently three types of container control supported by Natural:

- Group frames that have the (new) ’container’ style set. This can be changed in the dialog editor (via its attributes window) after the group frame has been created. If a group frame is converted to a container, all controls that are spatially contained within it are moved in the control hierarchy to become descendants of the group frame. If a group frame is converted to a non-container, all direct children of the group frame are moved up a level in the hierarchy to become siblings of the group frame.
- ActiveX controls which are marked as "OLEMISC_SIMPLEFRAME" in the registry. This flag is fixed by design for a particular ActiveX control class.
- Control boxes. This control type is always a control container. Indeed, that is its entire purpose in life. See the section "Working with Control Boxes" for more information.

Creating a nested control

Nested controls are created in the dialog editor in the same way as non-nested controls are. If, during control insertion, the initial left mouse button click is determined to be over a container control, the new control is created automatically by Natural as a child of that container. Even before the mouse button is clicked in insert mode, the dialog editor's status bar is continually updated with the container-relative mouse co-ordinates as the mouse cursor traverses the dialog.

In addition, nested controls can be indirectly created within the dialog editor when converting group frames to containers as described above.

At runtime, nested controls can be created dynamically, via the PROCESS GUI ACTION ADD statement for the nested control, by specifying the PARENT attribute as the handle of the required container control instead of the handle of the dialog. The nested control's position (RECTANGLE-X and RECTANGLE-Y attributes) should be specified relative to the container's client area. The client area of a control is the internal area of a control, excluding frame components such as 3-D borders, single-pixel frames resulting from used of the 'Framed' style, and a control's scroll bars.

Multiple selection, control sequence and clipboard operations

The dialog editor prohibits selection of multiple controls which do not have the same parent (i.e., are not all siblings of each other). This applies regardless of whether multiple controls are selected via "rubber banding" (marking of a region with the left mouse button held down) or via extended selection (holding down the <Shift> key whilst selecting a control). However, if a selected container control is deleted, then all its direct and indirect children (*descendants*) are of course implicitly deleted also, even though they are not explicitly selected. For this reason, a clipboard cut operation always copies the selected control(s) AND all descendant controls (if any) to the clipboard. For a clipboard copy operation, it is not clear whether to copy the container alone, or the container plus all its descendants. In this case, a message box is displayed, allowing the user to choose between the two options.

The pasting of controls from the clipboard uses the same control sequence (tab order) insertion position logic as for a control created from scratch. In both cases, the new control is created at a position in the control sequence immediately following the selected sibling (if any) plus any of its successive descendants. If a control other than a sibling is selected, an "effective sibling" is used instead, based on the position of the (active) selected control in the control sequence. The "active" selected control is the selected control (if any) which is highlighted using black (rather than gray) selection handles. If no selection is active, the control is inserted into the control sequence immediately preceding the first sibling control, or immediately after its container (or at the front of the control sequence for top-level controls) if the container is empty. Note, however, that the control sequence is maintained independently of the hierarchy. After a control has been created, it is possible to explicitly move any control to any position in the control sequence via the Control Sequence option on the Dialog menu.

The position of the newly-created control in the hierarchy is determined slightly differently in these two cases. In the case of a control being created from scratch, the container is determined by searching for the (topmost) container at the position where the left mouse button was pressed. However, in the case of pasting from the clipboard, we have no (X, Y)-position which we can use. In this case, the container is assumed to be the container of the selected control(s), or the dialog itself if no controls are selected. This means that if, for example, it is desired to copy and paste a control from one container to another, a control within the second container must be selected prior to the paste, not the container itself. If the second container is empty, this requires temporary creation of a dummy child control first, which can be deleted after the paste operation is complete.

Deletion of controls also deletes any of their descendant controls.

With the introduction of nested controls, the 'Select All' command has been changed to operate in the following manner:

- If no control is currently selected, the command selects all top-level controls.
- Otherwise, all other controls that are siblings of the currently selected one(s) are additionally selected.

Thus, in the common case where only one level of hierarchy is in use, the ’Select All’ command continues, as before, to select all dialog controls.

Working with a Dynamic Information Line

Event-driven applications are much more user-friendly when text in the dynamic information line (DIL) explains the dialog element that currently has the focus. A dialog element has the focus if it can receive the end user’s keyboard input.

You have two options to relate a dialog element to a DIL text:

- Use the dialog editor (most likely because it is the easiest way); or
- use Natural code to specify everything dynamically.

When you use the dialog editor, you will have to go through the following steps:

1. Set the attribute HAS-DIL to TRUE for the dialog by marking the "Dyn. Info Line" entry in the "Dialog Attributes" window.
2. Set the attribute DIL-TEXT to ’*diltextstring*’ for the dialog element. Push the "Source..." button to the right of the "DIL Text:" entry in the attributes window. The window "Specify attribute Source" appears. Choose one of the attribute sources and enter the text in the "Value" field. Ensure that ’*diltextstring*’ explains the dialog element’s usage in a short phrase.

When you use Natural code, the above two steps may look like this:

Example:

```
...
PERSDATA-DIALOG.HAS-DIL := TRUE  /* Set HAS-DIL To TRUE
#PB-1.DIL-TEXT := ’DILTEXTSTRING’   /* Assign the text string
...
```

Note: The STATUS-TEXT and the DIL-TEXT are displayed in the same area if the dialog has a status line and a text is displayed on the DIL.

Working with a Status Bar

In a similar way as the dynamic information line, the status bar makes an event-driven application more user-friendly.

The programmer has two options to relate a dialog element to a status bar:

- use the dialog editor (most likely because it is the easiest way); and
- use Natural code to specify everything dynamically.

When you use the dialog editor, you will have to:

- Set the attribute HAS-STATUS-BAR to TRUE for the dialog by marking the "Status Bar" entry in the "Dialog Attributes" window. The HAS-STATUS-BAR attribute determines whether the status bar may be modified. If HAS-STATUS-BAR is false, but HAS-DIL is true, the status bar appears, but is only used as dynamic information line.

When you use Natural code, the above step may look like this:

Example:

```
...
PERSDATA-DIALOG.HAS-STATUS-BAR := TRUE  /* Set HAS-STATUS-BAR To TRUE
PERSADTA-DIALOG.STATUS-TEXT := 'HELLO'  /* Set the text to 'Hello'
...
```

Note: The STATUS-TEXT and the DIL-TEXT are displayed in the same area if the dialog has a status line and a text is displayed on the DIL.

Working with Status Bar Controls

Note:
Status bar controls are not to be confused with the traditional dialog status bar which is created by selecting the 'status bar' check box in the Dialog Atttributes window in the Dialog Editor, or by setting the dialog's HAS-STATUS-BAR attribute at run-time. If you are using status bar controls, you should leave the 'status bar' checkbox unchecked and not set the HAS-STATUS-BAR attribute.

Creating a Status Bar Control

Status bar controls are created in the dialog editor in the same way as other standard controls (such as list boxes or push buttons) are. That is, they are either created statically in the Dialog Editor via the Insert menu or by drag and drop from the Insert tool bar, or dynamically at run-time by using a PROCESS GUI ACTION ADD statement with the TYPE attribute set to STATUSBARCTRL.

Unlike most other control types, status bar controls cannot be nested within another control and cannot be created within an MDI child dialog. In an MDI application, the status bar control(s) must belong to the MDI frame dialog.

A status bar control may have zero or more panes associated with it. Panes may be defined in the Dialog Editor from within the status bar control's attribute window, or at run-time by performing a PROCESS GUI ACTION ADD statement with the TYPE attribute set to STATUSBARPANE.

Using status bar controls without panes

A status bar control without panes offers restricted functionality, because most attributes providing access to the enhanced functionality of status bar controls are only supported for status bar panes. If you wish to do more with a status bar control than simply display a line of text, but don’t need to split up the status bar control into multiple sections, you should create a single pane that occupies the full width of the status bar control.

Stretchy vs. non-stretchy panes

If panes are defined for a status bar control, it should be decided whether each pane should stretch (or contract) when the containing dialog is resized, or whether it should maintain a constant width. The former are referred to here as ’stretchy’ panes, and the latter as ’non-stretchy’ panes.

There is no explicit flag in the Status Bar Control Attributes window to mark a pane as stretchy or non-stretchy. Instead, any pane defined with a width (RECTANGLE-W attribute) of 0 is implicitly assumed to be a stretchy pane, whereas any panes with a non-zero width definition are implicitly assumed to be fixed-width panes of the specified width (in pixels). Because the RECTANGLE-W attribute defaults to 0, all panes are initially stetchy when defined in the Dialog Editor.

The width of a visible stretchy pane is determined by taking the total width available for all panes in the status bar control, subtracting the widths of all visible fixed-width panes, then dividing the result by the number of visible stretchy panes.

Note:
The total available width for all panes normally excludes the sizing gripper, implying that the last pane stops short of the gripper, if present. However, if the status bar control has exactly one pane, and that pane is a stretchy pane, the full width of the dialog (including any sizing gripper) is used.

Outputting text to a status bar control

Text can be output to the status bar control in one of three ways:

1. For status bar controls with panes, by setting the STRING attribute of the pane whose text is to be set.
2. By setting the STRING attribute of the status bar control itself, which is equivalent to setting the STRING attribute of the first stretchy pane (if any) for status bar controls with panes.
3. By setting the STATUS-TEXT attribute of the dialog. This is equivalent to setting the STRING attribute of the status bar control (if any) identified by the dialog’s STATUS-HANDLE attribute.

Note that the last method is often the most convenient for setting the message text, because it does not require a knowledge of the status bar control or pane handles.

Example:

```
DEFINE DATA LOCAL
01 #DLG$WINDOW  HANDLE OF WINDOW
01 #STAT-1      HANDLE OF STATUSBARCTRL
01 #PANE-1      HANDLE OF STATUSBARPANE
END-DEFINE
...
#DLG$WINDOW.STATUS-HANDLE := #STAT-1
...
#PANE-1.STRING := ’Method 1’
...
#STAT-1.STRING := ’Method 2’
...
#DLG$WINDOW.STATUS-TEXT := ’Method 3’
```

Note:
The Dialog Editor automatically generates code to set the STATUS-HANDLE attribute to the first status bar control (if any). Therefore, the STATUS-HANDLE attribute only needs to be set explicitly if you are dynamically creating status bar controls, or if you have defined more than one status bar control in a dialog, and wish to switch between them.

Sharing a status bar in an MDI applications

Because status bar controls cannot be created for MDI child dialogs, it is convenient to not have to define multiple status bar controls in the MDI frame dialog. An alternative method is to define just a single status bar, and share it between each child dialog. This can be achieved as follows:

1. Define all possible panes you wish to use in your application within a single status bar control in the MDI frame dialog.
2. Mark all panes as ’shared’.
3. Export the handles of all panes to corresponding shadow variables in a GDA, so that the MDI child dialogs can access them directly.
4. In the COMMAND-STATUS event handler, set the VISIBLE attribute of all panes you wish to display for that dialog to TRUE. All other panes will be automatically made invisible.

Note:
In the COMMAND-STATUS event, you must also set the ENABLED state of any commands (signals, or menu or tool bar items which do not reference another object via their SAME-AS attribute) associated with the dialog, otherwise they will be automatically disabled. The commands associated with the dialog are all non-shared commands for the MDI frame and all shared commands for the active MDI child (or MDI frame, if no MDI child dialog is active).

Pane-specific context menus

Context menus are defined for the status bar control and not per-pane. However, if you wish to ensure that the context menu for a status bar control only appears when the user right clicks a particular pane, you can associate a context menu with the status bar control, but suppress it if the user clicks outside that pane.

Example:

```
DEFINE DATA LOCAL
01 #CTXMENU-1   HANDLE OF CONTEXTMENU
01 #STAT-1      HANDLE OF STATUSBARCTRL
01 #PANE-1      HANDLE OF STATUSBARPANE
01 #PANE-2      HANDLE OF STATUSBARPANE
01 #PANE-3      HANDLE OF STATUSBARPANE
01 #PANE        HANDLE OF STATUSBARPANE
01 #X (I4)
01 #Y (I4)
END-DEFINE
...
#STAT-1.CONTEXT-MENU := #CTXMENU-1
...
DECIDE ON FIRST *CONTROL
   ...
   VALUE #CTXMENU-1
       DECIDE ON FIRST *EVENT
        ...
    VALUE ’BEFORE-OPEN’
    /* Get click position relative to status bar control
    PROCESS GUI ACTION INQ-CLICKPOSITION WITH
      #STAT-1 #X #Y GIVING *ERROR
       /* Get pane (if any) at specified position
       PROCESS GUI ACTION INQ-ITEM-BY-POSITION WITH
      #STAT-1 #X #Y #PANE
    /* Only show context menu if user clicked in second pane
       IF #PANE = #PANE-2
    #CTXMENU-1.ENABLED := TRUE
       ELSE
    #CTXMENU-1.ENABLED := FALSE
    END-IF
     ...
 END-DECIDE
 ...
END-DECIDE
...
END
```

Note:
If you wish to display a different context menu for different status bar panes, the menu items must be created dynamically in the context menu’s BEFORE-OPEN Event.

Working with Dynamic Information Line and Status Bar

When you are working with both a Dynamic Information Line (DIL) and a status bar, the combination of the HAS-DIL and HAS-STATUS-BAR attributes determines whether and when DIL-TEXT and STATUS-TEXT values will be displayed:

HAS-DIL	HAS-STATUS-BAR	DIL-TEXT	STATUS-TEXT
TRUE	TRUE	displayed	displayed
TRUE	FALSE	-	-
FALSE	TRUE	-	displayed
FALSE	FALSE	-	-

If HAS-DIL and HAS-STATUS-BAR are TRUE, the DIL-TEXT will overlap the STATUS-TEXT value and vice versa, depending on which was modified last.

Adding a Maximize/Minimize/System Button

To add a Maximize/Minimize/System button to your dialog

Open the "Dialog Attributes" window. Check the "System Button" or "Maximizable"or "Minimizable" entry.

When the "System Button" entry is checked, the dialog’s standard control menu is available. This includes the control-menu box (to close the dialog), the title bar, and the Maximize and Minimize buttons.

Defining Color

You can define colors for dialogs and dialog elements. These can be foreground colors and background colors. To do this, you use the following attributes:

- BACKGROUND-COLOUR-NAME
- BACKGROUND-COLOUR-VALUE
- FOREGROUND-COLOUR-NAME
- FOREGROUND-COLOUR-VALUE

You can assign only standard colors to the attributes ending with NAME. The attributes ending on VALUE, however, can be assigned customized colors following the RGB model.

You can set colors:

- in an attributes window, or
- in event-handler code.

You can directly assign a value to the attributes ending with NAME. If you want to assign a value to an attribute ending with VALUE, you must set the NAME attribute to the value CUSTOM. If you do not set the NAME attribute to the value CUSTOM, the VALUE attribute is ignored.

Examples:

```
#DIA.BACKGROUND-COLOUR-NAME:= MAGENTA     /* Assign a value to a NAME
                                          /* attribute
#DIA.BACKGROUND-COLOUR-NAME:= CUSTOM      /* Set NAME to CUSTOM
#DIA.BACKGROUND-COLOUR-VALUE:= H’FF0000’  /* Then assign Red, Green, and
                                          /* Blue values to the VALUE
                                          /* attribute (hexadecimal)
```

Note: You can not use all customized colors in all parts of the user interface. Colors in text, for example, must always be monochrome.

When setting a color in an attributes window, you have three possibilities:

- Use the attribute ending with NAME and leave the value at DEFAULT. You can also do this in code. Your color will then be determined by your color settings in the windowing system.
- Use the attribute ending with NAME by pulling down the list-box and choose one of the predefined colors.
- Define your own color by using the attribute ending with VALUE.

To define a color

1. Click on the "Custom" push-button control right of the "Background color" entry.
   A dialog box appears.
2. Select one of the predefined colors or click on the "Define Custom Colors" push-button control. To set the red, green, and blue values, use the cursor to select the desired color or enter a value from 1 to 253 in the

red, green, and blue value display fields.

3. Click on the "Add to Custom Color" push-button control. To save the newly defined color, click on the OK button in the dialog box.
   The newly defined color is now selected by default.
4. To set it, close the attributes window.

Adding Text in a Certain Font

► To choose a specific font for the text assigned to a dialog element (for example, the caption on a push-button control), you use the dialog element’s attributes window

1. Click on the "..." push-button control to the right of the "Font" entry.
   A dialog box opens.
2. From the list of available fonts, select a font type, for example "Times New Roman".
3. From the list of styles available for the font type, select a font style, for example *"italics"*.
4. From the list of sizes available for the font type and style, select a font size, for example "10".
   A sample of your selected font will be displayed.
5. To set it: Close the attributes window.

Note: When adding centered or right-aligned text in a dialog element, the following minimum heights of the dialog element apply (RECTANGLE-H attribute): 4-point font - height of 8; 8-point - 22; 12-point - 24.

Additionally, the dialog editor allows selecting a font for the whole dialog in the dialog attributes window. This font is defined in the FONT-STRING attribute and is valid for the dialog and each of its children. A major advantage of selecting a font for the whole dialog is that if the chosen font is too large or too small for the dialog layout, you change the FONT-STRING attribute once instead of going through all children of the dialog.

Initially, the FONT-STRING attribute must be set as a parameter while the dialog is being created with PROCESS GUI action ADD. If a dialog element inside the dialog contains text with no particular font assigned to it, this text will be displayed in the font specified by FONT-STRING.

For more information on the FONT-STRING attribute and the way its value must be specified, see the Natural Dialog Components Manual.

Adding Online Help

From an application written with the dialog editor, you can invoke help for a specified help topic ID. Please bear in mind that you will have to create parts of the help associated with these help topic IDs outside the Natural development environment. You will also have to compile the help with the platform-specific help compiler.

To keep an overview of all the different help sections in an application, Natural provides you with the help organizer. With this organizer,

- you assign a help ID (HELP-ID attribute value) to a specific dialog element;
- you write the help text for the associated help topic; this text is converted to a .rtf file to be processed by the help compiler;
- you optionally define the help topic's keywords;
- you optionally assign a help compiler macro to the help topic;
- and optionally you add a comment for your internal documentation purposes.

To create a help topic

1. Invoke the help organizer's main dialog.
2. Select a particular dialog element.
3. Generate a new help topic ID.
4. Return to the help organizer main dialog.
5. Assign the generated help topic ID.
6. Enter the external definitions for the help topic ID, such as the help topic text and the topic name.
7. Return to the help organizer main dialog.
8. Go to the topic list and see whether this new help topic fits your general organization of the help file to be created.
9. Return to the help organizer main dialog.
10. Save everything.

A dialog or dialog element can also be assigned a HELP-ID number independently of the help organizer.

To do so

Open the corresponding attributes window. Enter a numeric value in the "Help ID" entry.

You must use the help topic's .h file to map the numerical ID that you enter here to the corresponding help topic ID (created by a markup in the .hlp file).

Natural expects the help file to be located in the resource (RES) subdirectory of the current library or one of the STEPLIBs, or in the directory referred to by the environment variable NATGUI_BMP. By default, Natural searches for a help file with the same name as the current library, but you can explicitely set the name of the help file via the HELP-FILENAME attribute.
If no file extension is specified, Natural searches for a compiled HTML help file with the extension ".chm" first, then (if not found) for a WinHelp help file with the extension ".hlp".
Thus, if no file extension is specified, it is possible to upgrade from using WinHelp to using HTML help without changing the Natural program. Note, however, that the Help Organizer only supports WinHelp. If you wish to create HTML help content, you must use an external help authoring tool to do so.

Whenever an end user presses F1 in an active dialog, Natural first queries for a file with the value of the HELP-FILENAME attribute plus the extension ".h*nn*" where *nn* is the Natural language code. If it does not find such a file, it queries for a file with the value of HELP-FILENAME plus the extension ".hlp"

Whenever the dialog element has the focus and the end user presses F1, Natural jumps to this help ID.

Note: When adding online help to an application, it is recommended to assign a HELP-ID number to each dialog and to write help texts for the dialogs. If the end user selects a dialog element to which no HELP-ID was assigned and presses F1 to request help, help on the current dialog will come up. If no HELP-ID was assigned to a dialog element, Natural will check whether the dialog element's parent - the dialog - has a HELP-ID. If not, Natural will check whether the dialog's parent - the dialog one level higher - has a HELP-ID, and so on, until the top-level dialog is reached.

To build a help file

1. Go to your command promt.
2. Change to the directory referred to by the environment variable $NATGUI_BMP.
3. Issue the command "HCRTF -X *helpfilename*".

Note:
This assures that the directory containing HCRTF.EXE is specified in the PATH environment variable.

To test a help file

1. Invoke a dialog in your application.
2. Press F1.

The help topic for the dialog should appear.

Alternatively, the help file can be conveniently built and tested interactively by opening the .hpj file in the Help Compiler Workshop (HCW.EXE).

To display help in a popup window

1. Check the Popup Help option in the dialog attributes window.
2. Run the dialog.
3. Press F1 with the focus on a control which has a help ID associated with it.

The help topic associated with the focus control should appear in a popup window.

Defining Mnemonic and Accelerator Keys

There are two ways of providing keyboard commands:

- A mnemonic key is determined by an underlined character in a visible dialog element, for example a menu item. The end user can select the menu item by pressing ALT+mnemonic key, for example ALT+A.
- An accelerator key is defined in the ACCELERATOR attribute. By pressing this key, the end user causes a double-click or click event for the dialog element regardless of whether the dialog element is visible or not, as long as the dialog element is enabled.

Defining a Mnemonic Key

You define a mnemonic key in the dialog element’s STRING attribute by specifying "&" before the desired character. At runtime, the character will be underlined. Example: the STRING attribute value "E&xplanation" will be displayed as "Explanation" at runtime.

If you define a mnemonic key with a text constant control or a group frame control, and the end user presses the mnemonic key at runtime, the next dialog element in the control sequence will get the focus. For example, if the next dialog element after a text constant control is an input-field control, the text constant control’s mnemonic key sets the focus to the input-field control. Whenever you disable such an input-field control at runtime, you should also disable the corresponding text constant control.

You can define mnemonic keys in the STRING attribute of the following types of dialog elements: group frame control, menu item, push-button control, radio-button control, text-constant control, toggle-button control, toolbar item.

You can still display an "&" in your runtime STRING by specifying "&&". Example: "A&&B" will be displayed as "A&B".

Note:
In recent Windows versions (e.g. Windows 2000), mnemonic characters are, by default, not underlined until the <Alt> key is pressed. However, this new behavior can be disabled by the user, such that mnemonic characters are always underlined. For example, this can be achieved on the English version of Windows 2000 by unchecking the

> "Hide Keyboard navigation indicators until I use the Alt key"

option under:

> "Start/Control Panel/Display/Effects."

Defining an Accelerator Key

You define an accelerator key by setting the ACCELERATOR attribute to a key or a key combination for the dialog element, for example to "F6" or "CTRL+1". If the end user presses the accelerator key, the double-click event occurs for the dialog element, or if no double-click event is available, the click event occurs. The accelerator key does not work if the corresponding event is suppressed, or if the dialog element is disabled.

Standard system accelerators such as "Alt+Esc", "Ctrl+Esc", "Alt+Tab" and "Ctrl+Alt+Del" can be defined as accelerators, but do not cause the dialog element’s click or double-click event to be triggered. Instead, they cause the associated system functionality to be invoked. The same applies to standard MDI accelerators (such as "Ctrl+F4" and "Ctrl+F6") if used within MDI applications and to any accelerators belonging to in-place activated servers (e.g. ActiveX controls which currently have the focus).

Note that user-defined accelerator keys overwrite identical user-defined shortcut keys associated with desktop items.

If the same accelerator key is associated with more than one dialog element, the dialog element whose click or double-click event is triggered is not defined.

A dialog element which references another via its SAME-AS attribute inherits the accelerator of the referenced object. For example, if a menu item references a signal, and the signal’s accelerator is "Ctrl+Alt+X", then querying the menu item’s ACCELERATOR attribute will also return "Ctrl+Alt+X". However, the accelerator, if pressed, will only trigger a click event for the referenced dialog element (i.e., the signal in this example).

Accelerators of the form "Alt+X", where "X" is one of the alphabetic characters, should be avoided, because they are "reserved" for use as keyboard mnemonics.

Displaying Accelerator Keys in Menus

In order to show accelerators for menu items, the menu text needs to first be appended with a tab (h’09’) character and then appended with the text for the accelerator. This cannot be done statically in the dialog editor’s menu editor, because there is no way to enter a tab character into the string definition. However, the accelerators may be appended dynamically using a generic piece of code which iterates round all menu items for a dialog. This is illustrated by the following external subroutine, which can conveniently be called from within a dialog’s AFTER-OPEN event.

Example:

```
 DEFINE DATA
 PARAMETER
   1 #DLG$WINDOW  HANDLE OF WINDOW
 LOCAL
   1 #CONTROL     HANDLE OF GUI
   1 #COMMAND     HANDLE OF GUI
 LOCAL USING NGULKEY1
 END-DEFINE
 *
 DEFINE SUBROUTINE APPEND-ACCELERATORS
 #CONTROL := #DLG$WINDOW.FIRST-CHILD
 REPEAT UNTIL #CONTROL = NULL-HANDLE
   IF #CONTROL.TYPE = SUBMENU OR #CONTROL.TYPE = CONTEXTMENU
  #COMMAND := #CONTROL.FIRST-CHILD
  REPEAT UNTIL #COMMAND = NULL-HANDLE
 IF #COMMAND.ACCELERATOR <> ’ ’
     COMPRESS #COMMAND.STRING H’09’ #COMMAND.ACCELERATOR INTO
 #COMMAND.STRING LEAVING NO SPACE
 END-IF
    #COMMAND := #COMMAND.SUCCESSOR
  END-REPEAT
  END-IF
  #CONTROL := #CONTROL.SUCCESSOR
END-REPEAT
END-SUBROUTINE
END
```

This dynamic technique has the advantage that the accelerator does not, in effect, have to be defined twice (i.e., for the ACCELERATOR and STRING attributes of the menu item).

Note that if the target language is not English, the ACCELERATOR attribute value will probably have to be translated before being appended to the menu item string.

Dynamic Data Exchange - DDE

Concepts

DDE is a protocol defined by Microsoft Corp. to enable different applications to exchange data. This means that, for example, an application written in Natural may exchange data with a spreadsheet, because they are both able to process the DDE protocol. An application that processes the DDE protocol communicates with another DDE application via standardized messages. One of the applications is defined as the client, the other as the server. Client and server are holding a DDE conversation.

Note: For an overview of DDE concepts and terminology, see your Microsoft Windows documentation.

Data in a DDE conversation is identified by a three-level hierarchy:

- service,
- topic,
- item.

A DDE conversation is established whenever a client requests a *service* from a DDE server. A DDE server offers one or more *services* to all active applications.

For each service, a DDE server may offer any number of *topics*. The DDE client then requests a conversation on a *topic* of a *service*.

In a conversation on a *topic* of a *service*, the DDE client and the DDE server uniquely identify data to be exchanged by an *item* name.

A DDE server may support a number of services, which in turn may consist of a number of topics, which themselves may contain a number of items.

With Natural, you can develop both DDE client applications as well as DDE server applications. You may, for example, write a Natural DDE client application that requests data from a spreadsheet acting as a DDE server, or you may write a Natural DDE server application that supplies a word processor (DDE client) with data.

To develop DDE client and DDE server applications, the following functionality is provided:

- A number of NGU-prefixed subprograms in library SYSTEM; these send messages and data as defined in the parameter data area "NGULDDE1"
- a parameter data area (NGULDDE1) which describes the parameters used by the subprograms in a DDE conversation (the "DDE-VIEW");
- a DDE-Client event and a DDE-Server event which handle DDE messages.

You develop a DDE server application by reacting to the DDE-Server event and by using the NGU-SERVER-prefixed subprograms from library SYSTEM to register services and topics and to send messages and data to the DDE client application.

You develop a DDE client application by reacting to the DDE-Client event and by using the NGU-CLIENT-prefixed subprograms from library SYSTEM to initiate conversations and send requests and other DDE commands to DDE server applications.

You always have to include the parameter data area NGULDDE1 and the local data area NGULFCT1 in your client or server dialog. (You need NGULFCT1 in order to use the NGU-prefixed subprograms in library SYSTEM).

Developing a DDE Server Application

Registering/Unregistering Services and Topics

Before a DDE server application can be addressed by a DDE client application, it must register its service names and all supported topics for the services. You use subprogram NGU-SERVER-REGISTER to do this for each service/topic the DDE server supports. Registering will usually be handled in the "after open" event of the base dialog.

When registering a service/topic for the first time, you will need to supply Natural with the dialog-ID of the dialog that will function as the server and that will therefore receive all DDE messages from clients. This is done by setting the DDE-VIEW.CONV-ID to the respective dialog-ID and also by setting DDE-VIEW.MESSAGE to the string ’DLGID’.

Note that at a later time you are able to add more topics to a service or even entirely new services. You can also make a topic unavailable by using subprogram NGU-SERVER-UNREGISTER.

Getting Data From The Client

After successful registration, it is possible that the DDE server application receives DDE messages from a DDE client application which is establishing a conversation on a registered topic of a service.

Such messages for a DDE server are received in the DDE-Server event of the dialog. At the beginning of the event-handler section, it is necessary to fill the DDE-VIEW with the client’s message data. This is done by using subprogram NGU-SERVER-GET-DATA. After reading the data, it will be necessary to act based on the client message received. The possible messages and their meaning are explained in the description of subprogram NGU-SERVER-GET-DATA.

Sending Data To The Client

In many cases, the client message ultimately requires the server to send data to the client. This is achieved by using the subprogram NGU-SERVER-DATA.

Terminating DDE Server Operation

Whenever DDE server operation is supposed to terminate, you use the subprogram NGU-SERVER-STOP. It unregisters all services and terminates all active conversations. You terminate the server application with the CLOSE DIALOG statement.

Developing a DDE Client Application

Connecting With The DDE Server Application

In order to establish a conversation with a DDE server application, a DDE client application must call the subprogram NGU-CLIENT-CONNECT with the service and topic name of the server it wants to connect. In order to receive the appropriate DDE events from a server, it is necessary to set the DDE-VIEW.CONV-ID to the client’s dialog-ID and also to set DDE-VIEW.MESSAGE to the string ’DLGID’. The call will return a unique conversation ID in DDE-VIEW.CONV-ID. This value must be set appropriately in all further communication with the server.

Using The Services of a DDE Server Application

The client has several options to use the services of a server once a conversation has been established. It can

- request data on a specific item (using NGU-CLIENT-REQUEST),
- send data to the server (using NGU-CLIENT-POKE),
- ask the server to execute a command (using NGU-CLIENT-EXECUTE), or

- establish a warm or hot link to the server (using NGU-CLIENT-ADVISE-HOT, NGU-CLIENT-ADVISE-WARM and NGU-CLIENT-ADVISE-TERM).

Receiving Data From The DDE Server Application

The DDE client will receive data or other messages from the DDE server via the client dialog’s DDE-Client event. Whenever a server has sent a message, this event occurs. The message contents must first be retrieved using NGU-CLIENT-GET-DATA. This will fill the DDE-VIEW structure appropriately. The client must then determine which message (DDE-VIEW.MESSAGE) has arrived and react appropriately. The possible messages are listed in the description of subprogram NGU-CLIENT-GET-DATA.

Disconnecting From The DDE Server Application

Whenever the client determines that the conversation is no longer needed, a call to NGU-CLIENT-DISCONNECT must be issued to inform the server that the conversation is to be terminated.

Terminating DDE Client Operation

Whenever the client application terminates or wants to stop using DDE, it needs to call NGU-CLIENT-STOP. This informs Natural to close all active conversations of the client and shut down DDE operation for the application.

Return Codes

Possible return codes are described in this section:

Note: Each error-code description is not necessarily comprehensive. In these cases, the description is marked with an asterisk (*).

Code	Meaning
-1	You have specified an incorrect command or command parameter. Ensure that your DDE data area is of the correct type and that the command is correct.
0	The function was processed correctly.
1	This value is returned when an application has attempted to initialize with the DDEML library more than once. Check the logic of your program. Also ensure that the DDEML was exited correctly during the last run of the program.
2	This value may be returned from the server-initialize function if you have run the program before and not exited the DDEML correctly. It is also returned by a call-back function, whenever the requested service failed.
	An error occurred in the underlying layer.*
3	The conversation ID referenced does not represent an active conversation. Check if you have specified a correct service name.
4	The application could not initialize with the DDE library as the maximum number of instances are connected.
5	The DDEML communication has not been initialized. You must initialize with the DDEML before any DDE activity can take place.
6	Memory allocation problems encountered. This error might occur if the queue of messages for either part in the conversation becomes too long. *

Code	Meaning
7	A service, topic or item name was longer than 255 characters. Check if your fields are correctly specified for DDE-VIEW and make sure that you are not attempting to place a string longer than 255 characters in any one of the above variables.
8	An error occurred in the DDE library. Contact SOFTWARE AG Support.*
9	Parameters passed to this function were illegal. This can be returned by any function call. Check your parameters.
10	"Server Type Link" is supported but no call-back function for UNLINK is passed to the function "PIDsRegisterTopic". *
11	An attempt was made to remove a topic for which at least one conversation is still active. This includes trying to unregister a topic for which a conversation still exists.
12	The service/topic referenced has not been registered with the function "PIDsRegisterTopic".
13	No links were active for the DDE-VIEW.SERVICE when the NGU-Server-Data subprogram was used. Check your service name and use the DDE-SPY in the SDK Tool Kit to see what services are available.
14	The requested type of link is invalid.
15	The transaction ID is corrupted. Check the value of your transaction ID in your DDE view.
16	The client application requested a conversation and prior to that, no function was specified to send the data for the links.
17	An asynchronous transaction was requested, but the client application did not specify a function to send details of the completed transaction. Such a function must be specified when the conversation is initialized.
18	A synchronous transaction timeout expired. The amount of time taken for your transaction to complete was longer than the TIMEOUT value in your DDE-VIEW structure. Increase the TIMEOUT value or set it to "-1" for indefinite waiting.
19 - 24	For internal use only.

Object Linking and Embedding - OLE

What Is OLE In The Natural Context?

Natural supports the following OLE technologies:

- OLE Documents
- OLE Visual Editing (In-place Activation)
- ActiveX Controls

If you are new to OLE, it is highly recommended that you first get a basic overview by referring to one of the various sources available. One such source, for example, is the Microsoft Win32 software development kit documentation.

OLE Documents Support

OLE documents is a technology that integrates different Windows applications seamlessly so that the end user can concentrate on the data rather than on handling the different applications. With OLE you can, for example, embed a Word for Windows document in a Natural dialog. Whenever the end user enters the text container to edit the document, the entire Word functionality is available. Thus, the end user does not have to invoke Word.

OLE Documents Support is provided by the Natural dialog element OLE container control. For more information, see the section The OLE Container Control.

The OLE documents technology defines container and server applications. A container application is an application that is able to use objects created by a server application. These objects are used by linking or embedding them. In this context, Natural is the container application because the dialog editor provides an OLE container control. A typical server application is Microsoft Word; the Word documents would then be the objects used by Natural.

Embedding and Linking

- Linking means that the content of a document is accessed via a link to an external file. This file is stored in the server’s format (for example, a file in ".rtf" format would be stored in a file system outside Natural; the server residing in this external file system would be Microsoft Word).
- Embedding means that the content of a document is maintained in the container application and is stored in the container’s internal format. Embedded documents are created
  - either by building them from scratch in the container application;
  - or by loading an external document.

Embedded objects are edited by visual editing ("in-place activation"), whereas linked objects must be opened in an extra server window for editing.

Natural provides the dialog element OLE container control for embedding and linking documents. Furthermore, Natural provides actions to save and load embedded documents in internal Natural format. By default, these embedded objects in internal format are stored and retrieved in the %NATGUI_BMP% directory with a default extension of ".neo" (Natural Embedded Object).

If you want the OLE container control to display an embedded object when the dialog starts

1. Invoke the container control’s attribute window.
2. Set the Type entry to "Existing OLE Object".
3. Select a file specification in the Name field.

If you want to display an embedded object dynamically at runtime

Use the PROCESS GUI statement action OLE-READ-FROM-FILE (see also Dialog Components Manual, Chapter Executing Standardized Procedures).

If you want the OLE container control to display a linked object when the dialog starts

1. Invoke the container control’s attribute window.
2. Set the Type entry to "OLE Server".
3. In the "Select OLE Server or Document" dialog that comes up, select "Create From File" and select a file specification.

If you want to display a linked object dynamically at runtime

Assign the file specification of the external document to the attribute SERVER-OBJECT of the OLE container control (see also Dialog Components Manual, Chapter Attributes).

Visual Editing - In-place Activation

In-place activation means that the end user is able to activate a server application in the container application’s window. Such a server application is related to an object embedded in a Natural dialog’s OLE container control. The server application is activated by double-clicking on the OLE container control. The Natural dialog’s toolbar and menu-bar control are then merged with the server application’s menu and toolbar. The dialog now contains toolbar items and menu items that enable you to edit the object with the help of the server’s functionality.

ActiveX Controls Support

ActiveX controls support enables the Natural programmer to use the many third-party ActiveX controls inside a Natural dialog. Natural enables you to access the ActiveX controls properties and methods direct and to program the ActiveX controls events.

ActiveX controls support is provided by the Natural dialog element "ActiveX control". For more information, see Working with ActiveX Controls.

The OLE Container Control

Creating an OLE Container Control

You can create an OLE container control either statically in the dialog editor or dynamically at runtime.

Creating an OLE Container Control in the Dialog Editor

The OLE container control enables you to integrate server applications. You can integrate server applications in the following three ways, as indicated by the "Object Information" group frame, "Type" entry of the OLE container control’s attributes window.

- Type: New OLE object. You create an OLE container control that acts as a placeholder for the insertable object. At runtime, your end user can create the embedded object by starting the server application. The embedded object can then be saved as Natural embedded object (.neo file).
- Type: Existing OLE object. Your end user changes an existing embedded object in the OLE container control. The embedded object is saved as Natural embedded object (.neo file).
- Type: OLE server. You create a native OLE object in your application or you create a link to an external object.

To create an OLE container control in the dialog editor

1. In the dialog editor main menu, choose "Insert", then "OLE Container".
2. Draw a rectangle by holding down the right mouse button, dragging the mouse vertically/horizontally and releasing the mouse button.

An empty OLE container is created.

To display a document in the OLE container when starting the dialog

1. Double-click the OLE container control to invoke the attribute window.
2. In the "Type" selection box, choose "OLE server" for linking an external document.
   Or choose "Existing OLE object" for reading in an embedded object.
3. Press the "..." button to select the external or embedded object file.

Creating an OLE Container Dynamically At Runtime

Before you enter the examples in an event-handler section, declare a handle variable for the OLE container control in the local data area of the dialog:

```
01 #OCT-1 HANDLE OF OLECONTAINER
```

Example for creating an OLE container control at runtime and linking an external document:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #OCT-1
   TYPE = OLECONTAINER
   SERVER-OBJECT = ’PICTURE.BMP’
   RECTANGLE-X = 56
   RECTANGLE-Y = 32
   RECTANGLE-W = 336
   RECTANGLE-H = 160
   PARENT = #DLG$WINDOW
   SUPPRESS-CLICK-EVENT = SUPPRESSED
   SUPPRESS-DBL-CLICK-EVENT = SUPPRESSED
   SUPPRESS-CLOSE-EVENT = SUPPRESSED
   SUPPRESS-ACTIVATE-EVENT = SUPPRESSED
   SUPPRESS-CHANGE-EVENT = SUPPRESSED
END-PARAMETERS GIVING *ERROR
```

Example for creating an OLE container control at runtime and embedding a Natural embedded object:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #OCT-1
   TYPE = OLECONTAINER
   EMBEDDED-OBJECT = ’SLIDE.NEO’
   RECTANGLE-X = 56
   RECTANGLE-Y = 32
   RECTANGLE-W = 336
   RECTANGLE-H = 160
   PARENT = #DLG$WINDOW
   SUPPRESS-CLICK-EVENT = SUPPRESSED
   SUPPRESS-DBL-CLICK-EVENT = SUPPRESSED
   SUPPRESS-CLOSE-EVENT = SUPPRESSED
   SUPPRESS-ACTIVATE-EVENT = SUPPRESSED
   SUPPRESS-CHANGE-EVENT = SUPPRESSED
   END-PARAMETERS GIVING *ERROR
```

Clearing or Deleting an OLE Container At Runtime

This section contains examples for clearing and deleting an OLE container at runtime.

Before you enter the examples in an event-handler section, declare a handle variable for the OLE container control in the local data area of the dialog:

```
01 #OCT-1 HANDLE OF OLECONTAINER
```

Example for clearing (removing the document of) the OLE container control:

```

PROCESS GUI ACTION CLEAR WITH #OCT-1
```

Example for deleting the OLE container control:

```
PROCESS GUI ACTION DELETE WITH #OCT-1
```

OLE Container Controls And The Dialog’s Menu Bar

The menu item attribute MENU-ITEM-OLE can have four different values which detemine if and where the menu item in question is displayed during in-place activation of a server (see also Dialog Components Manual, Chapter Attributes).

The menu item attribute MENU-ITEM-TYPE also has the value MT-OBJECTVERBS. This enables you to have the OLE container control display the available server actions (command verbs) in this menu item (see also Dialog Components Manual, Chapter Attributes).

Other OLE Container Control Functionality

While a document is displayed in an OLE container control, the end user has the possibility to activate the default command verb of the server by double-clicking inside the OLE container control’s rectangle. This is equivalent to executing the PROCESS GUI statement action OLE-ACTIVATE (see also Dialog Components Manual, Chapter Executing Standardized Procedures). Furthermore, the end user can select a server command verb by displaying a popup menu. You display this popup menu by holding down the right mouse button inside the OLE container. Then you select the desired command verb and release the mouse button.

If the MODIFIABLE attribute of an OLE container control is set to FALSE, a double-click on the container does not start the default command verb of the server and holding down the right mouse button does not show the popup menu with the available server command verbs (see also Dialog Components Manual, Chapter Executing

Standardized Procedures).

During visual editing (in-place activation), the server uses the Natural dialog for the editing of the document. The server does its work as a task on its own and the Natural processing continues. Thus, it is possible to execute event code and, for example, to limit the visual editing to a certain time by specifying PROCESS GUI ACTION OLE-DEACTIVATE, WITH #OCT-1 in a timer's event section (see also Dialog Components Manual, Chapter Executing Standardized Procedures).

Attributes, Events and PROCESS GUI Statement Actions

The following sections list all the attributes, events and PROCESS GUI statement actions that apply specifically to the OLE container control.

Attributes

The OLE-specific attributes provided with the OLE container control are:

- EMBEDDED-OBJECT (see Dialog Components Manual, Chapter Attributes)
- ICONIZED (see Dialog Components Manual, Chapter Attributes)
- OBJECT-SIZE (see Dialog Components Manual, Chapter Attributes)
- SERVER-OBJECT (see Dialog Components Manual, Chapter Attributes)
- SERVER-PROGID (see Dialog Components Manual, Chapter Attributes)
- SUPPRESS-ACTIVATE-EVENT (see Dialog Components Manual, Chapter Attributes)
- SUPPRESS-CLOSE-EVENT (see Dialog Components Manual, Chapter Attributes)
- ZOOM-FACTOR (see Dialog Components Manual, Chapter Attributes)

Event

This OLE-specific event occurs when a server application is activated:

- Activate event (see Dialog Components Manual, Chapter Events)

PROCESS GUI Statement Actions

The OLE-specific PROCESS GUI statement actions provided with the OLE container control are:

- OLE-ACTIVATE (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-DEACTIVATE (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-GET-DATA (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-INSERT-OBJECT (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-READ-FROM-FILE (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-SAVE-TO-FILE (see Dialog Components Manual, Chapter Executing Standardized Procedures)
- OLE-SET-DATA (see Dialog Components Manual, Chapter Executing Standardized Procedures)

SYSMAIN Utility - Overview

The following topics are covered below:

- SYSMAIN Command
- Copy Object(s)
- Available XREF Options
- Move Object(s)
- Rename Object(s)
- Delete Object(s)
- List an Object
- Find Object(s)
- Importing Object(s) to a Library
- Invoking SYSMAIN by a Subprogram
  - Direct Commands
  - Additional Keywords for Direct Commands

SYSMAIN Command

The SYSMAIN command is used to perform operations Natural objects such as copy, move, delete or import. In most cases this can be accomplished using drag and drop or cut, copy and paste objects (see Object Operations) within the Natural Studio environment. Also the import of objects can be done with this technique.

But one can still imagine some situations where more detailed preselections are necessary to perform an object operation (e.g copy only objects with a specific date) which cannot be covered using drag and drop or cut, copy and paste.
Additionally this utility can be used as a system command inside a Natural application. See also Invoking SYSMAIN by a Subprogram.

For all object operations, the command SYSMAIN is used to display the starting dialog.

Copy Object

1. Select the "Copy" radio button and choose OK.
   The "Copy" dialog box is displayed.
2. From the "Library" list box, select the source library.
3. If the source library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" box, enter a string pattern, including one or more wildcard characters to filter the objects to be displayed for selection. The default is " * ", all objects.
5. In the "Type" group box, select the types of objects to be copied.
   If you select Programming, you can further limit the types of programming objects to be copied by pressing the "Object Types" button. The default is all object types.
6. Select "Source" and/or "Cataloged" to copy either the source or the cataloged object module, or both.
7. For an explanation of the options available with "XREF", see Available XREF Options.
8. To display only objects for selection last saved by a particular user, enter the user ID. The default is no user ID.
9. To display only objects for selection saved up to a particular point in time, enter a cut-off date and time. All objects up to and including this time are displayed for selection. The default is date 0000-00-00, time 00:00.
10. From the "Library" list box, select the target library.
11. If the target library is not in the current file, modify the database ID (DBID) and file number (FNR).
12. In the "Name" box, enter a new string pattern, including one or more wildcard characters to rename the objects selected in the filter specified above. The string cannot contain more characters than the source string. The default is "*".
    The "Confirm on Replace" toggle button is selected by default. A warning message will appear if you attempt to copy an object to another library where an object exists with the same name. (The object in the destination library is replaced.) You can override the warning by turning the "Confirm on Replace" function off.
    If you want to give the copied object a different name, select the "Rename Object(s)" option.
13. Choose OK.
    A list box is displayed containing all objects selected in the "Copy" dialog box.
14. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
15. Choose "Copy" to copy the objects.
    If the "Rename Object" option is active, the "Rename" dialog box is displayed. You can enter a new name for the copied object. If you want to skip the current object and display the name of the next selected object, choose "Skip".

Available XREF Options

NO	Cross-reference data are not processed, except when using the DELETE function. If a cataloged object is deleted, SYSMAIN always deletes any existing XREF data for this object.
YES	All cross-reference data are processed.
FORCE	All cross-reference data are processed and the object must be documented in Predict.
DOC	All cross-reference data are processed and the object must be documented in Predict but XREF data are not copied.

Move Object

1. Select the "Move" radio button and choose OK.
   The "Move" dialog box is displayed.
2. From the "Library" list box, select the source library.
3. If the source library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" box, enter a string pattern, including one or more wildcard characters to filter the objects to be displayed for selection. The default is "*", all objects.
5. In the "Type" group box, select the types of objects to be moved.
   If you select Programming, you can further limit the types of programming objects moved by pressing the "Object Types" button. The default is all object types.
6. Select "Source" and/or "Cataloged" to move either the source or the cataloged object module, or both.
7. To display only objects for selection last saved by a particular user, enter the User ID of the user who last saved the objects to be moved. The default is no User ID.
8. To display only objects for selection saved up to a particular point in time, enter a cut-off date and time. All objects up to and including this time are displayed for selection. The default is: date 0000-00-00, time 00:00.
9. From the "Library" list box, select the target library.
10. If the target library is not in the current file, modify the database ID (DBID) and file number (FNR).
11. In the "Name" box, enter a new string pattern, including one or more wildcard characters to rename the objects selected in the filter specified above. The string cannot contain more characters than the source string. The default is "*".
    The "Confirm on Replace" toggle button is selected by default. A warning message will appear if you attempt to move an object to another library where an object exists with the same name. (The object in the destination library is replaced.) You can override the warning by turning the "Confirm on Replace" function off. If you want to give the moved object a different name, select the "Rename Object(s)" option.
12. Choose OK.
    A list box is displayed containing all objects selected in the "Move" dialog box.
13. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
14. Choose "Move" to move the objects.
    If the "Rename Object" option is active, the "Rename" dialog box is displayed. You can enter a new name for the moved object. If you want to skip the current object and display the name of the next selected object, choose "Skip".

Rename Object

1. Select the "Rename" radio button and choose OK.
   The "Rename" dialog box is displayed.
2. From the "Library" list box, select the source library.
3. If the source library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" box, enter a string pattern, including one or more wildcard characters to filter the objects to be displayed for selection. The default is "*", all objects.
5. In the "Type" group box, select the types of objects to be renamed.
   If you select Programming, you can further limit the types of programming objects renamed by pressing the "Object Types" button. The default is all object types.
6. To display only objects for selection last saved by a particular user, filter the objects to be displayed for selection, enter the User ID of the user who last saved the objects to be renamed. The default is no User ID.
7. To filter the objects to be displayed for selection, enter a cut-off date and time. All objects up to and including this time are displayed for selection. The default is: date 0000-00-00, time 00:00.
8. In the "Name" box, enter a new string pattern, including one or more wildcard characters to rename the objects selected in the filter specified above. The string cannot contain more characters than the source string. The default is "*".
9. Select "Source" and/or "Cataloged" to rename either the source or the cataloged object module, or both.
   The "Confirm on Replace" toggle button is selected by default. A warning message will appear if you attempt to use the name of an existing object in the same library. You can override the warning by turning the "Confirm on Replace" function off.
10. Choose OK.
    A list box is displayed containing all objects selected in the "Rename" dialog box.
11. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
12. Choose "Rename" to rename the objects. For each object selected, a rename window is displayed.
13. In the successive rename windows, enter the new names for the objects.

Delete Object

1. Select the "Delete" radio button and choose OK.
   The "Delete" dialog box is displayed.
2. From the "Library" list box, select the source library.
3. If the source library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" box, enter a string pattern, including one or more wildcard characters to filter the objects to be displayed for selection. The default is "*", all objects.
5. In the "Type" group box, select the types of objects to be deleted.
   If you select Programming, you can further limit the types of programming objects deleted by pressing the "Object Types" button. The default is all object types.
6. To display only objects for selection last saved by a particular user enter the User ID of the user who last saved the objects to be deleted. The default is no User ID.
7. To display only objects for selection saved up to a particular point in time, enter a cut-off date and time. All objects up to and including this time are displayed for selection. The default is date 0000-00-00, time 00:00.
8. Select "Source" and/or "Cataloged" to delete either the source or the cataloged object module, or both. The "Confirm on Delete" function is selected by default.
9. Choose Object List.
   A list box is displayed containing all objects selected in the "Delete" dialog box.
10. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
11. Choose "Delete" to delete the objects.

List an Object

1. Select the "List" radio button and choose OK.
   The "Object Maintenance - List" dialog box is displayed.
2. From the "Library" list box, select the library.
3. If the library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" box, enter a string pattern, including one or more wildcard characters to filter the objects to be displayed for selection. The default is "*", all objects.
5. In the "Type" group box, select the types of objects to be listed.
   If you select Programming, you can further limit the types of programming objects copied by pressing the "Object Types" button. The default is all object types.
6. Select "Source" and/or "Cataloged" to list either the source or the cataloged object module, or both.
7. To display only objects for selection last saved by a particular user, enter the User ID. The default is no User ID.
8. To display only objects for selection saved up to a particular point in time, enter a cut-off date and time. All objects up to and including this time are displayed for selection. The default is date 0000-00-00, time 00:00.
9. Choose Object List.
   A list box is displayed containing all objects selected in the "Object Maintenance - List" dialog box.
10. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
11. 11. Choose OK

Find Object

1. Select the "Find" radio button and choose OK.
   The "Find Object" dialog box is displayed.
2. From the "Library Name" list box, select the library to be searched. Enter "*" to search through all libraries.
3. If the search library is not in the current file, modify the database ID (DBID) and file number (FNR).
4. In the "Name" text box, enter the search string to be used. The default is "*", all objects.
5. In the "Type" group box, select the types of objects to be found.
   If you select Programming, you can further limit the types of programming objects found by pressing the "Object Types" button. The default is all object types.
6. To search for objects last saved by a particular user, enter the User ID. The default is no User ID.
7. To display only objects saved up to a particular point in time, enter a cut-off date and time. All objects up to and including this time are displayed. The default is: date 0000-00-00, time 00:00.
8. Choose "Object List".
   If more than one library is specified in the "Library Name" list box, a dialog box is displayed containing a list of all libraries which meet the search criteria. Otherwise, go to Step 10.
9. From the list box, select one or more libraries to be searched for the object and choose OK.
   The system searches all libraries for objects that meet the search criteria, and for each library searched, places a list of objects found in the list box. From this dialog, you can list the objects found.
10. Choose "Select All" to select all objects in the list box, or select objects individually with the mouse pointer.
11. Choose "List" to list the objects.

Importing Objects to a Library

Use the import function whenever you need to copy objects from an external source to a Natural library. When you import objects, the target library’s file directory FILEDIR.SAG is automatically updated to contain information on the newly imported objects. Be aware that if you use other copy utilities (such as the Windows File Manager) to copy objects to a Natural library, the file directory will not be updated and you can not access the objects from that library.

Notes:
Within Natural it is possible to define object names with a "#" or a "+" character. Those objects cannot be imported with the SYSMAIN utility, because the names, starting with "#" or "+" are encrypted in the FILEDIR.SAG and SYSMAIN does not look into FILEDIR.SAG when importing objects. Use the Natural Object Handler LOAD/UNLOAD command, because the Natural Object Handler is able to handle objects with such a specific name.
If an object is imported and the object name is unknown to Natural and exists in the library, a container name will be generated with the object name identical plus a running index.

To import an object:

1. Select the "Import" radio button and choose OK.
   The "Import Object" dialog box is displayed.
2. In the "Path" combo box of the "Source" group frame, either enter the path for the source library directly in the text box or choose the path from the list box. A likely path might look like the following:
   C:\NATAPPS\FNAT\SYSTEM\gp
3. In the "Name" text box, enter the string to be used to preselect objects from the source library. The default is the asterisk "*", which includes all objects. Note that this is not the final selection method.
4. In the "Code" group frame, check "Source" and/or "Cataloged" to specify which types of code for of each object are to be imported. The default is both source code and cataloged code. One type must be selected.
5. In the "Target" group frame, in the "Library" drop-down combo box, choose the name of the library to which the objects are to be imported, or if you want to create a new library, in the text box, enter the new library name.
6. In the "Target" group frame, enter a database ID (DBID) and file number (FNR) for the target library.

7. In the "Target" group frame, in the "User ID" field, enter a User ID if necessary.
8. In the "Mode" group frame, select the option button "Report or Structure" to specify whether the imported objects are to be used in report or structured mode.
9. In the "Confirm on Replace" toggle button, specify whether you want to be prompted when an object in the target library has the same name as an object in the source library.
10. Choose Object List or press ENTER.
11. A second "Import Object" dialog box is displayed with a list of objects fulfilling the criteria specified in the "Source" group frame of the previous dialog.
12. Select the individual objects to be imported or choose "Select All" to select all of the objects in the window.
13. Choose "Import" to import the selected objects to the target library. The objects are imported and the dialog boxes are closed automatically.

Invoking SYSMAIN by a Subprogram

MAINUSER is a subprogram which allows you to perform the various SYSMAIN functions directly from any user-written object (subroutine, program or subprogram) without going through the normal steps of invoking SYSMAIN. Upon completion of processing of the SYSMAIN functions, the utility is terminated and control is returned to the object from which the request was issued. MAINUSER can be used in either online or batch mode.

MAINUSER is invoked as follows:

CALLNAT ’MAINUSER’ ***command error message library***

The parameters are:

command (A250)	The direct command string to be executed by SYSMAIN.
error (N4)	The return code issued by SYSMAIN at the end of processing to indicate a normal end of processing or an error.
message (A72)	The message corresponding to the error given online.
library (A8)	The name of the library containing the utility SYSMAIN; by default, this is the library SYSMAIN. (Under UNIX, OpenVMS and Windows this parameter does not apply and is provided for compatibility reasons only.)

An example of a callable routine is program MAINCALL in library SYSMAIN.

Direct Commands

SYSMAIN functions can be executed using direct commands issued as a parameter of the MAINUSER subprogram.

Direct commands consist of keywords and parameters. The sequence of the direct command syntax is not completely fixed. The rules which apply are:

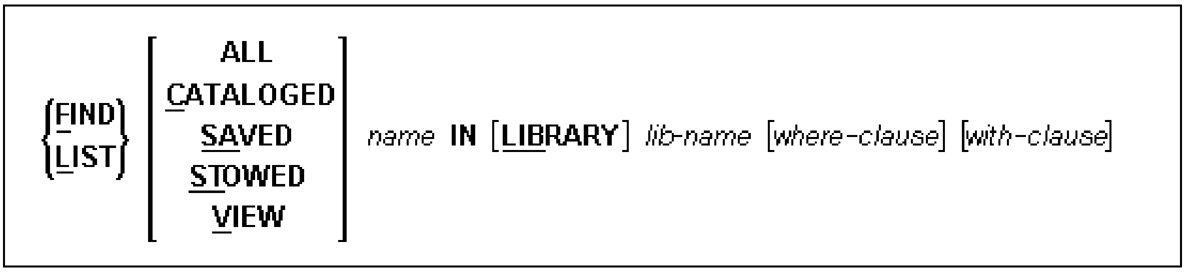
- Function, object type and object name must be the first three parameters of the command string.
- The library or path name must be specified immediately after the FROM, IN and TO keywords. (If the optional keyword LIBRARY or PATH is used, it must be entered between the FROM, IN or TO keyword and the library or path name).
- The WHERE clause must always follow the FROM, IN or TO keyword and the library name; the sequence of the keywords and values within the clause can be specified in any order.
- The keywords and values of the WITH clause can be specified in any order, but the WITH clause must

always be placed at the end of the command string.

Note:
In the syntax diagrams below, FM is shown instead of FROM to make the diagrams easier to read; however, FROM can always be used as a synonym for FM and vice versa.

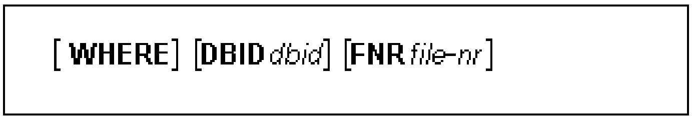
FIND and LIST Direct Command Syntax

The direct command syntax of the FIND and LIST functions is:



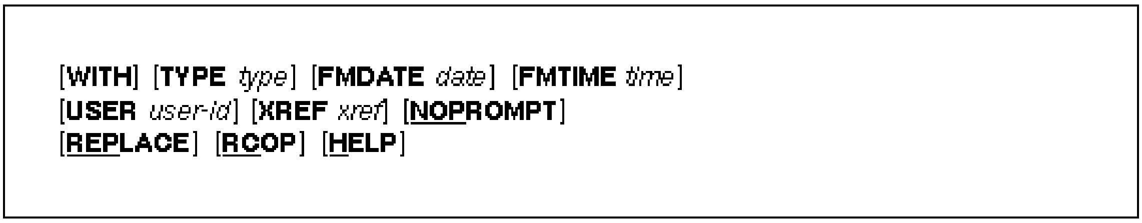
```
          ALL
        CATALOGED
{FIND}    SAVED      name IN [LIBRARY] lib-name [where-clause] [with-clause]
{LIST}    STOWED
          VIEW
```

The *where-clause* is optional. The syntax is:



```
[WHERE] [DBID dbid] [FNR file-nr]
```

The *with-clause* is optional. The syntax is:



```
[WITH] [TYPE type] [FMDATE date] [FMTIME time]
[USER user-id] [XREF xref] [NOPROMPT]
[REPLACE] [RCOP] [HELP]
```

Examples:

COPY *PROG1* FM *TESTORD* TO *ORDERS* DBID *1* FNR *6* REP
C *PGM** WITH REP TYPE *PNS* FM *PRODLIB* TO *TESTLIB*

M *PROG1* TO *NEWLIB*
MOVE STOWED * TO *NEWLIB* WHERE DBID *100* FNR *160* FMDATE *87-11-01* FM *OLDLIB*
WITH XREF *Y*

Since the WHERE clause and WITH clause syntax are identical for each function, they are only shown once with the FIND and LIST command syntax above.

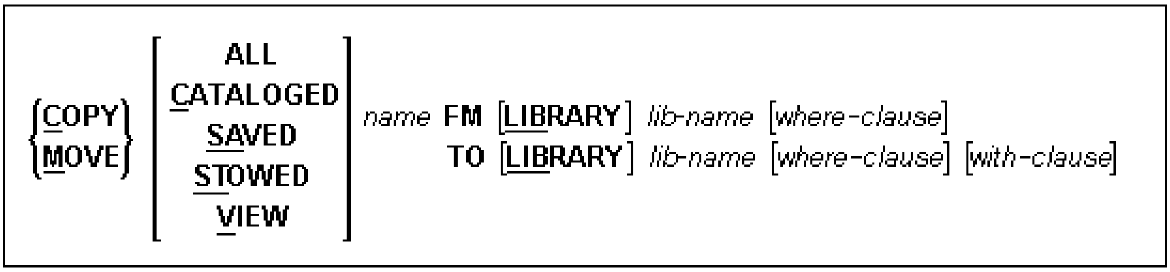
Examples:

FIND *PROG1* **IN DBID** *1* **FNR** *6*
FIND STOWED *MAINMENU* **IN** *SYS** **WHERE DBID** *1* **FNR** *5*
FIND ALL *PROG2* **IN** *PROD** **FNR** *27* **DBID** *1*

LIST VIEW * **IN** *lib-name*
L SAVED *TEST** **IN** *lib-name* **TYPE** *PNS* **FNR** *6*
L SA *TEST** **TYPE** *PM* **IN** *lib-name* **FNR** *6* **DBID** *2*

COPY and MOVE Direct Command Syntax

The direct command syntax of the COPY and MOVE functions is:



```
        ALL
{COPY}  CATALOGED  name FM [LIBRARY] lib-name [where-clause]
{MOVE}  SAVED           TO [LIBRARY] lib-name [where-clause] [with-clause]
        STOWED
        VIEW
```

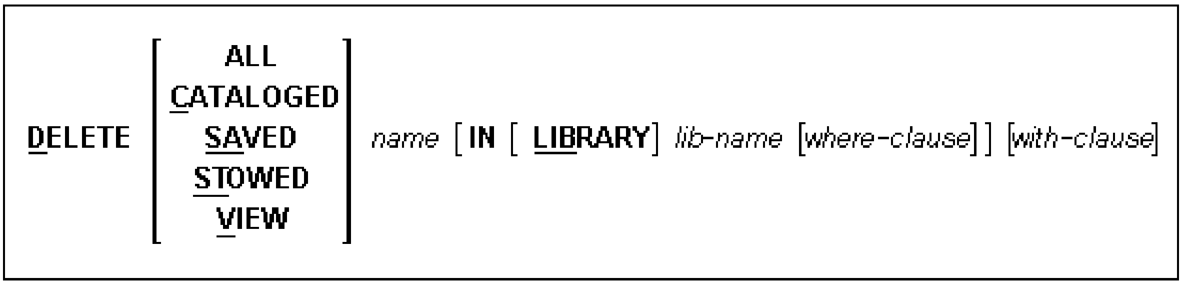
Examples:

COPY *PROG1* **FM** *TESTORD* **TO** *ORDERS* **DBID** *1* **FNR** *6* **REP**
C *PGM** **WITH REP TYPE** *PNS* **FM** *TESTLIB* **TO** *PRODLIB*
COPY VIEW *PERS* **FM** *OLDLIB* **FNR** *10* **TO** *NEWLIB* **FNR** *16* **REPLACE**

M VIEW *PERSONNEL* **FM** *OLDLIB* **FNR** *20* **TO** *NEWLIB* **FNR** *24*
M *PROG1* **TO** *NEWLIB*
MOVE STOWED * **FM** *OLDLIB* **WITH XREF** *Y* **TO** *NEWLIB* **WHERE DBID** *100* **FNR** *160*

DELETE Direct Command Syntax

The direct command syntax of the DELETE function is:



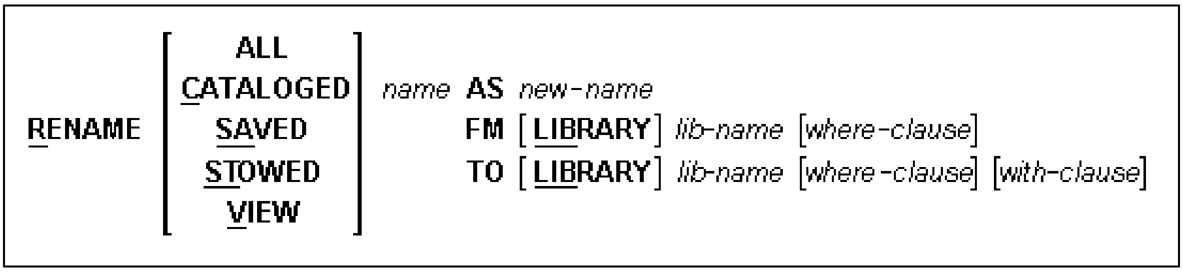
```
        ALL
        CATALOGED
DELETE  SAVED      name [IN [ LIBRARY] lib-name [where-clause]] [with-clause]
        STOWED
        VIEW
```

Examples:

D SA * **IN** *LIBTEST* **TYPE** *GLA*
DEL * **TYPE** *PM* **IN** *TESTORD*
DEL VIEW *FINANCE* **IN** *TESTLIB* **DBID** *12* **FNR** *27*

RENAME Direct Command Syntax

The direct command syntax of the RENAME function is:



```
        ALL
        CATALOGED  name AS new-name
RENAME  SAVED           FM [LIBRARY] lib-name [where-clause]
        STOWED          TO [LIBRARY] lib-name [where-clause] [with-clause]
        VIEW
```

Examples:

RENAME *PGM1* **AS** *PROG1*
REN *PGM1* **AS** *PROG1* **FM** *TESTLIB* **DBID** *1* **FNR** *5* **TO** *PRODLIB* **DBID** *2* **FNR** *6*

Additional Keywords for Direct Commands

In addition to the keywords shown with the parameters above, the following keywords can also be used with direct commands to specify selection criteria:

Keywords	**Explanation**
ALL	All saved and/or cataloged programming objects are selected for processing.
CAT	All cataloged programming objects are selected for processing. (Any corresponding saved programming object is not processed.)
HELP	Activates online help.
IN/FM	Refers to a source environment.
NOPROMPT	Suppresses all prompts.
RCOP	Used with direct commands to specify that a copy of the object being renamed is to be made.
SAVED	All saved programming objects are selected for processing. (Any corresponding cataloged object is not processed.)
STOWED	All programming objects which are both saved and cataloged are selected for processing.
TO	Refers to a target environment.
WITH	Optional keyword to indicate the start of a *with-clause*.
WHERE	Optional keyword to indicate the start of a *where-clause*.
.	End of command. If this character is detected anywhere within a command string, all subsequent data are ignored.

Tamino Server Extensions

The following topics are covered:

- Introduction
- Overview
- Developing a Tamino Server Extension
- Using Callbacks
- Deploying a Tamino Server Extension
- Installing a Tamino Server Extension
- Tamino Server Extension Exsample

Introduction

Tamino allows you to develop, implement, administrate and execute Server Extensions. Tamino Server Extensions can be used to extend the query and mapping Tamino Server functionality by adding user-defined logic. For a description of the functionality available with Tamino Server Extensions, see the Tamino documentation.

In order to extend the Tamino functionality, you install Tamino Server Extension Packages in Tamino databases. These packages contain (among other data) Tamino Server Extension Objects based on COM or on Java. Methods of these objects can then be used to extend the query or mapping functionality of the Tamino Server.

Server Extension Objects based on COM can be implemented in Natural. Please check the Natural Release Notes for information about the Tamino version needed as a prerequisite.

Overview

This document focuses on the Natural-specific implementation details of Tamino Server Extensions and the Natural tools and techniques used in this process. Essential background information that should help you develop valid Server Extension Function code is contained in the Tamino documentation. You should read the corresponding chapters of the Tamino documentation carefully before starting to develop Tamino Server Extensions.

- To develop Natural-based Server Extensions, you use the Natural Class Builder. A Tamino Server Extension is developed as a NaturalX class that implements interfaces corresponding to a predefined structure. To support the implementation of these interfaces, certain predefined Natural modules are delivered with Natural.
- To deploy a Natural-based Tamino Server Extension in the target environment, you use the usual Natural deployment tools.
- To register your Tamino Server Extension, you use the Natural REGISTER command.

Once you have developed, installed and registered your Natural-based Tamino Server Extension using Natural development tools, you can assign it to a Tamino database and use it in a Tamino schema using the usual Tamino tools. For a detailed description of the usage of these tools, see the Tamino documentation.

- To select methods of your Natural-based Tamino Server Extension into a Server Extension Package and to create a package file, you use the SXS Analyzer.
- To install and administrate Server Extensions in a Tamino database, use the Server Extensions Administration as provided by the Tamino Manager.
- To assign Server Extension functions to a Tamino schema, use the Tamino Schema Editor
- Server Extensions can be traced using the SXS Trace.

Developing a Tamino Server Extension

The following topics are covered.

- Overview
- Set the Library SYSEXSXS as Steplib
- Create a New Library for your Project
- Create a NaturalX Class
- Create the Object Data Area
- Edit the Object Data Area
- Link the Connection Interface
- Implement the Method Connect
- Add Server Extension Functions
- Save and Catalog the Class

Overview

The Natural Class Builder is used to develop a Tamino Server Extension. A Natural-based Tamino Server Extension is a NaturalX class that implements interfaces corresponding to a predefined structure. To support the implementation of these interfaces, certain predefined Natural modules are delivered with Natural:

- An Interface Module (Copycode) containing the declaration of the Connection interface defined by Tamino.
- Parameter Data Areas containing parameter definitions for the different types of Server Extension functions.

Set the Library SYSEXSXS as Steplib

When implementing a Tamino Server Extension in Natural, you can use a number of predefined Natural modules contained in the sample library SYSEXSXS. This makes sure that your Tamino Server Extension conforms to the interface defined by Tamino. In order to use these modules in your Tamino Server Extension project, first define the library SYSEXSXS as steplib.

To define the library SYSEXSXS as steplib:

1. Start the Natural Configuration Utility.
2. Select the Natural Parameter Module you are working with.
3. Choose Edit > Find to locate the parameter LSTEP.
4. Enter SYSEXSXS into the list of steplibs.
5. Save the Natural Parameter Module.
6. Close the Natural Configuration Utility.
7. Restart Natural Studio.

Create a New Library for Your Project

It is recommended to put all Natural modules for one Tamino Server Extension into one Natural library. Therefore first create a new Natural library for your project.

To create a new library:

1. Select "User Libraries" in the library workspace.
2. Select New in the context menu.
3. Choose a name for the library.

Create a NaturalX Class

A Tamino Server Extension is implemented as a NaturalX Class. Therefore create a new class.

To create a new class:

1. Select your library in the library workspace.
2. Select New Source > Class in the context menu.
3. Choose a name for the class.
4. Select Save in the context menu.
5. Choose a name for the class module.

Create the Object Data Area

An Object Data Area in a NaturalX class is used to contain variables that shall keep their value during the lifetime of an instance of the Tamino Server Extension.

To create an Object Data Area and link it to your class:

1. Select your class in the library workspace.
2. Select New > Object Data Area in the context menu.
3. Choose a name for the Object Data Area.

Edit the Object Data Area

The Object Data Area of a Tamino Server Extension should at least contain an object handle to hold a reference to the Tamino callback object during the lifetime of an instance of your Tamino Server Extension. The Tamino callback object is passed by Tamino to the Server Extension when it loads the Extension. You can use the methods of the callback object to call Tamino functionality from inside your Tamino Server Extension functions.

To add an object handle for the callback object to your Object Data Area:

1. Doubleclick the Object Data Area in the library workspace to edit it.
2. In the Data Area Editor select Insert > Handle.
3. Choose a name for the object handle (e. g. CALLBACK).
4. Select "Object" as handle type.
5. Click the Add button.
6. Click the Quit button.
7. If you so wish, add further variables to the Object Data Area as required by your Server Extension.
8. Close the Data Area Editor and save the Object Data Area.

Link the Connection Interface

A Tamino Server Extension must implement the Connection interface ISXSConn. This interface is declared in the interface module ICONN-C in the library SYSEXSXS. To be able to locate the interface module, you must have defined the library SYSEXSXS as steplib. Link this interface module to your class.

To link the interface module to your class:

1. Select your class in the library workspace.
2. Select Link > Interface Module in the context menu.
3. Select the interface module ICONN-C in library SYSEXSXS.

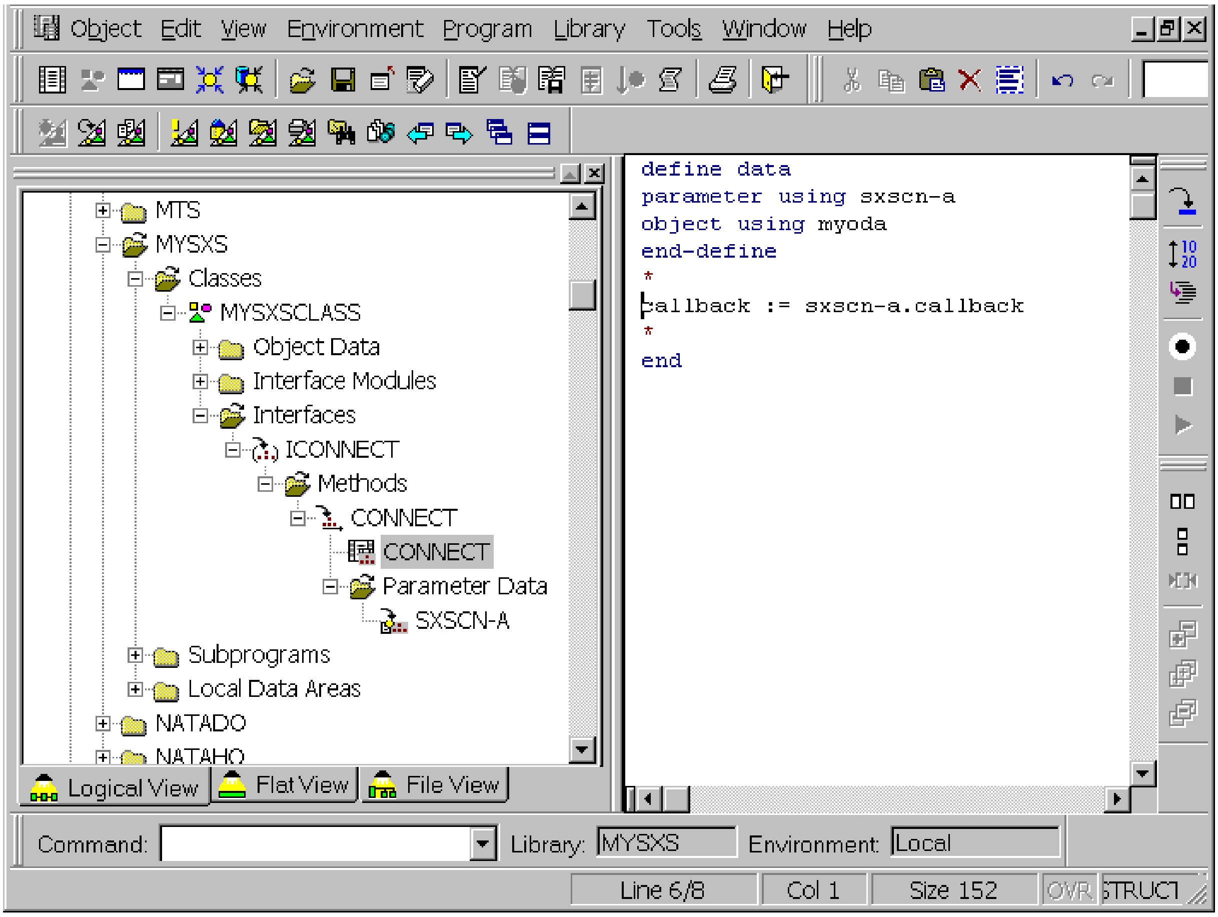
Implement the Method Connect

The method Connect of the ISXSConn should store a handle to the callback object in the Object Data Area. This allows the Server Extension Functions to access Tamino functionality.

To implement the method Connect:

1. Fully expand the branch "Interfaces" of your class in the library workspace.
2. Select the subprogram node "CONNECT". This is the default name for the subprogram that will implement the method Connect.
3. If you so wish, rename the subprogram to a name of your choice by selecting Rename in the context menu.
4. Doubleclick the subprogram node to create and edit the method subprogram.

The implementation of the Connect method must include both the Object Data Area of the class and the Parameter Data Area of the method Connect. The body of the method must assign the Callback object handle from the Parameter Data Area to the corresponding object handle defined in the Object Data Area, as shown in the example below.



```
define data
parameter using sxscn-a
object using myoda
end-define
*
callback := sxscn-a.callback
*
end
```

You can also add further initialization code to the method Connect as required by your Server Extension.

Add Server Extension Functions

You can now start to add your own Server Extension Functions to the class. First you will create a new interface for your class to contain the Server Extension Functions. Then you will add the individual functions to that interface.

To create a new interface:

1. Select the node "Interfaces" of your class in the library workspace.
2. Select New in the context menu.
3. Choose a name for the interface.

To create a new Server Extension Function:

1. Select your interface in the library workspace.
2. Select New > Method in the context menu.
3. Choose a name for the method.
4. Select the method
5. Select Link > Parameter Data Area.

Tamino distinguishes different types of Server Extension Functions. Depending on the type of Server Extension Function to be implemented, choose the corresponding Parameter Data Area:

Function	Data
Map In function:	Parameter Data Area SXSMI-A
Map Out function:	Parameter Data Area SXSMO-A
On Delete function:	Parameter Data Area SXSDL-A
Event function:	Parameter Data Area SXSEV-A
Query function:	Parameter Data Area SXSQU-A

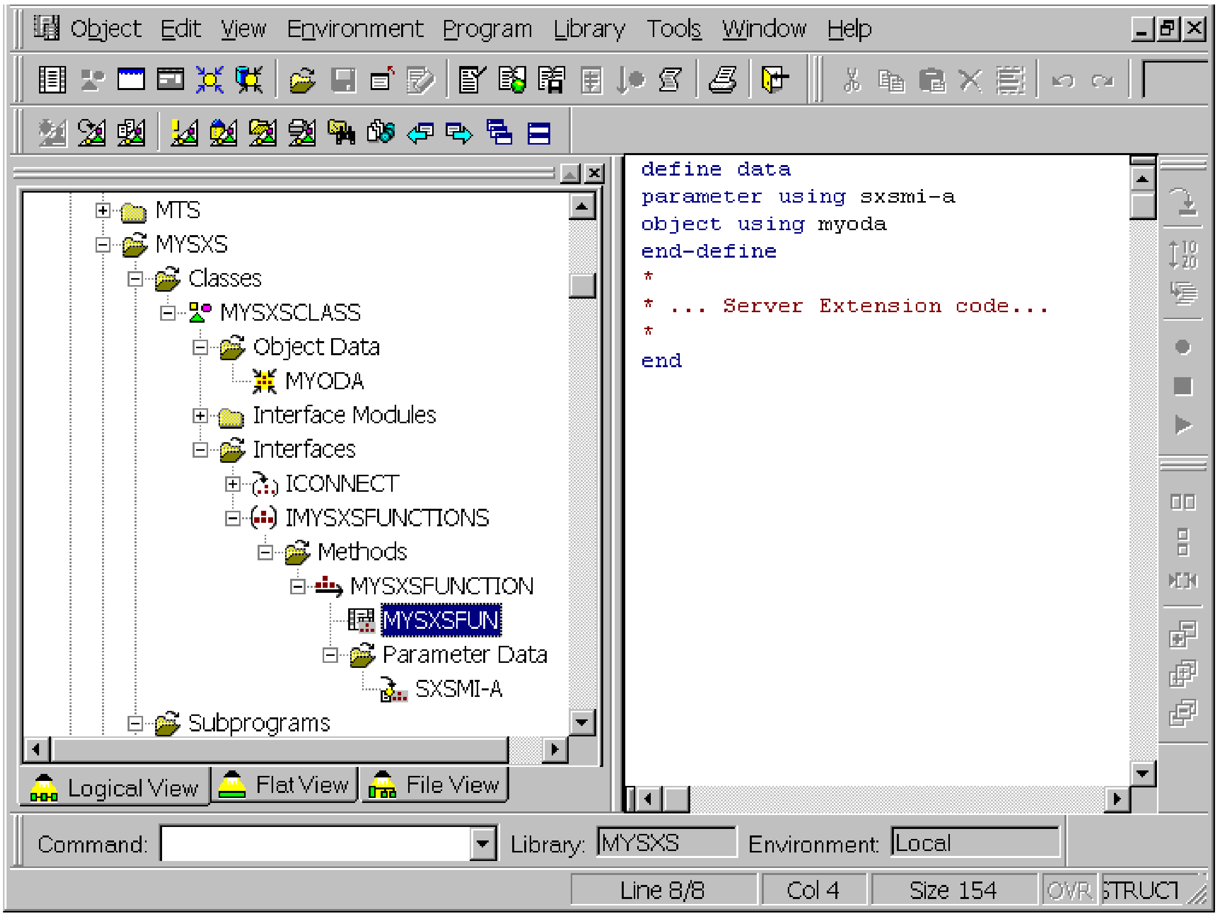
These Parameter Data Areas are contained in the library SYSEXSXS. To be able to locate the Parameter Data Areas, you must have defined the library SYSEXSXS as steplib.

Note:
The Parameter Data Area SXSQU-A is just an example. Query functions can have user-defined parameters. Thus it is not possible to define a common Parameter Data Area for them. If you want to create a query function, please consult the Tamino documentation and check which parameter types are allowed in query functions. Then create your own Parameter Data Area in your project library that matches the needs of your query function.

To implement the Server Extension Function:

1. Select the subprogram node that represents the method implementation.
2. If you so wish, rename the subprogram to a name of your choice by selecting Rename in the context menu.
3. Doubleclick the subprogram node to create and edit the method subprogram.

The implementation of the method must include both, the Object Data Area of the class and the Parameter Data Area assigned to the method. The body of the method contains the coding specific to the Server Extension Function.



```
define data
parameter using sxsmi-a
object using myoda
end-define
*
* ... Server Extension code...
*
end
```

- Close the Program Editor and save the subprogram.

To add further Server Extension functions, repeat " To create a new Server Extension Function ".

Save and Catalog the Class

Finally save the class and recatalog the whole project library.

1. Select your class in the library workspace.
2. Select Save in the context menu.
3. Select your library in the library workspace
4. Select Cat All in the context menu.

Using Callbacks

Tamino callbacks are interfaces of the Tamino Server that can be used in a Server Extension Function. They enable access to both the various databases that can be administrated by the Tamino Server and system information available in the running Tamino Server. To use a Callback function from within a Natural based Tamino Server Extension, do the following:

- Consult the Tamino documentation to find out the parameters of the callback function you wish to use.
- In the Object Data Area of the NaturalX class that implements your Tamino Server Extension you have defined an object handle as a reference to the callback object. Send a corresponding method call to this object handle.

Deploying a Tamino Server Extension

Having developed the NaturalX class that implements your Tamino Server Extension, deploy it into the target environment with the usual Natural deployment tools, for instance the Object Handler. A Tamino Server Extension must be installed on the same machine where the Tamino server is running.

Register the class in the target environment under an arbitrary COMSERVERID. If necessary, see the NaturalX documentation on COMSERVERIDs and the different options of the REGISTER command.

Installing a Tamino Server Extension

Installing a Natural-based Tamino Server Extension is the same procedure as installing any COM-based Tamino Server Extension:

1. Use the SXS Analyzer to create a Server Extension Package. In the SXS Analyzer, select as the file to analyze the type library of your NaturalX class. As with any NaturalX class, the type library is located in the directory <*natdir*>\<*natvers*>\Natural\Etc\<*comserverid*>\<*classname*>\<*version*>, where <*natdir*> is the Natural installation directory, <*natvers*> the installed Natural version, <*comserverid*> the COMSERVERID under which the class was registered, <*classname*> the class name and <*version*> the class version (currently always "v1"). Proceed as usual with the SXS Analyzer to create a Server Extension Package.
2. Install the Server Extension Package into a Tamino database using the Server Extensions Administration as provided in the Tamino Manager.
3. Afterwards you can use the Tamino Server Extension as usual, for instance in XQuery functions or to map XML sub-documents in the Tamino Schema Editor.

Tamino Server Extension Sample

The Natural sample library SYSEXSXS contains a simple Natural-based Tamino Server Extension as a programming example. The sample works on the Employees schema and maps parts of the schema, the salary data, on Server Extension Functions. These functions work as follows:

- Whenever an Employee element is inserted into the schema, the salary data of this Employee is passed to the Map In function SXSStoreToFile. This function creates a file in the TEMP directory and stores the data into the file.
- Whenever an Employee element is read from the schema, the salary data of this Employee is requested from the Map Out function SXSRetrieveFromFile. This function opens the corresponding file in the TEMP directory and reads the data from the file.
- Whenever an Employee element is deleted from the schema, the On Delete function SXSDeleteFile is called. This function deletes the corresponding file in the TEMP directory.

The sample Employees schema and some sample data are contained in the RES subdirectory of the sample library SYSEXSXS.

The sample can be driven comfortably using the Tamino DOM Demo application contained in the sample library SYSEXINO. To run the sample Tamino Server Extension, proceed as follows:

1. Install the sample Tamino Server Extension in a Tamino database as described above in "Deploying a Tamino Server Extension" and "Installing a Tamino Server Extension".
2. Start the dialog MENU in the library SYSEXINO.
3. Set the Tamino URL to your Tamino database.
4. Enter "NATSXSDemoData" as collection name.
5. Execute "Define Tamino Schema" and select the schema "EmployeeSXSSchema.tsd" from the RES subdirectory of the sample library SYSEXSXS.

6. Execute "Load Tamino Data" and select the data file "EmployeeSXSData.xml" from the RES subdirectory of the sample library SYSEXSXS.
7. Execute the sample queries and update records. Note that part of the Employee data (the salary data) is now handled by the Tamino Server Extension.
8. Delete the Employee data.
9. Delete the Employee schema.

XML Toolkit

The XML Toolkit enables developers to process XML documents within Natural.

The toolkit includes a wizard which generates Natural source code and provides the following features:

- Mapping Natural data definitions to DTDs;
- Serializing a Natural data structure and assigning its contents to an XML file;
- Mapping DTDs to Natural data definitions;
- Parsing an XML file and assigning its contents to a Natural data structure.

The wizard and the context-related help texts are included in the library SYSEXXT.